Titre: Title:	Plate-forme d'acquisition d'événements asynchrones générés par l'interface radio d'un commutateur de téléphonie mobile
Auteur: Author:	Bartosz Balazinski
Date:	2001
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Balazinski, B. (2001). Plate-forme d'acquisition d'événements asynchrones générés par l'interface radio d'un commutateur de téléphonie mobile [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/6969/

UNIVERSITÉ DE MONTRÉAL

PLATE-FORME D'ACQUISITION D'ÉVÉNEMENTS ASYNCHRONES GÉNÉRÉS PAR L'INTERFACE RADIO D'UN COMMUTATEUR DE TÉLÉPHONIE MOBILE

BARTOSZ BALAZINSKI
DÉPARTEMENT DE GÉNIE ÉLECTRIQUE
ET DE GÉNIE INFORMATIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE ÉLECTRIQUE)

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

PLATE-FORME D'ACQUISITION D'ÉVÉNEMENTS ASYNCHRONES GÉNÉRÉS PAR L'INTERFACE RADIO D'UN COMMUTATEUR DE TÉLÉPHONIE MOBILE

Présenté par: Bartosz Balazinski

En vue de l'obtention du diplôme de: Maîtrise ès sciences appliquées

A été dûment accepté par le jury d'examen constitué de:

M. Ettore Merlo, Ph.D., président

M. Jean-Louis Houle, Ph.D., membre et directeur de recherche

M. Michel Dagenais, Ph.D., membre

à Sonia

REMERCIEMENTS

Je tiens à remercier mon directeur de recherche, le professeur Jean-Louis Houle pour son aide, ses conseils, ses encouragements et sa patience. Je voudrais également remercier les professeurs Ettore Merlo et John Thistle pour leur aide et leurs conseils, ainsi que mes collègues de Ericsson Research Canada, Angelo Cuffaro et Robert Brunner, sans qui ce projet n'aurait pas eu lieu. Je voudrais aussi remercier Jean-François Bertrand pour sa contribution à l'idée originale et Dominique Hérvé pour son aide précieuse dans la révision du mémoire. Je ne voudrais surtout pas oublier de remercier ma famille et mes amis pour leur support tout au long de ce projet.

RÉSUMÉ

La téléphonie mobile du type TDMA («Time Division Multiple Access») est définie par plusieurs normes dont la norme nord-américaine ANSI-136. Un des éléments essentiels décrits par cette norme est l'interface radio. Plus particulièrement, il s'agit de l'ensemble des messages et des procédures qui permettent aux terminaux mobiles d'établir des connexions téléphoniques. La compagnie Ericsson propose une solution qui implante cette norme au sein de sa gamme de commutateurs numériques à usage général appelés AXE-10.

Dans notre projet de recherche, nous avons conçu le prototype d'une plate-forme flexible permettant l'acquisition des événements asynchrones. Ces événements résultent du fonctionnement de l'interface radio de l'AXE-10, c'est-à-dire, qu'ils sont générés par les différents modules du logiciel de contrôle du commutateur et leurs séquences reflètent directement l'architecture et le fonctionnement de l'implantation de la norme ANSI-136. Ils peuvent être observés et analysés afin de superviser ou optimiser l'efficacité de l'interface radio. Cependant, les événements individuels ne constituent que de faibles sources d'information. C'est pourquoi notre plate-forme doit reconstruire leur séquence pour chaque appel afin d'obtenir des informations intéressantes. L'objectif ultime de notre plate-forme est de servir de fondation pour des applications d'analyse avancée en temps «presque» réel de ces séquences d'événements.

Afin d'atteindre cet objectif le système conçu est composé de quatre éléments principaux. Premièrement, une interface TCP/IP est utilisée afin de permettre le transfert des événements à partir du commutateur vers un poste de travail externe sur lequel réside notre plate-forme. Deuxièmement, au sein de notre plate-forme, un décodeur permet de transformer les événements reçus en structures de données utilisées par le reste du

système ainsi que par les applications d'analyse. Troisièmement, à l'aide d'algorithmes de dispersion les événements sont triés selon une clé qui identifie les différents appels en cours. Finalement, les séquences d'événements sont reconstruites à l'aide d'un algorithme d'analyse syntaxique de type LR(0). Cet algorithme se base sur une grammaire fournie par l'usager qui décrit les règles de fonctionnement de l'interface radio du commutateur numérique.

Nous avons vérifié le fonctionnement de notre plate-forme par une analyse théorique et par une série de tests de performance avec des données réelles. Durant ces tests nous avons remarqué que le temps de traitement des événements individuels était pratiquement indépendant de la quantité de données présentes dans le système ainsi que leur taux d'arrivée. D'autre part, l'exécution en temps réel prend environ un dixième du temps disponible. Cela nous donne une bonne indication que notre système pourra servir comme base aux applications dont l'exécution doit se faire temps «presque» réel.

ABSTRACT

The TDMA ("Time division multiple access") mobile telephony systems are specified by various standards; the North American version of TDMA is specified by the ANSI-136 standard. One of the key parts described by that standard is the radio interface. In fact, among other elements, these specifications present a set of messages and procedures that allow mobile terminals to establish telephone connections. Ericsson Company implements ANSI-136 on its general-purpose AXE-10 digital switches.

In our project, we have developed and prototyped a flexible platform to acquire asynchronous events which are generated by the AXE-10 radio interface. These events are, in fact, generated by various modules of the controlling software. Their sequences represent the architecture and performance of the ANSI-136 implementation. These events may be analysed in order to monitor or optimise the radio interface performance. Taken separately, the amount of information they provide is very small. This is why our platform has to rebuild their sequences for each call in progress in order to extract information of interest. Our main goal is to provide a foundation for some advanced analysis applications running in "near" real-time.

In order to achieve this goal our system is divided into four main parts. The first part is the TCP/IP interface that allows extracting the asynchronous events from the AXE-10 and transferring them into a workstation where the main part of our platform is running. The second part of our system is a decoder that transforms the asynchronous events received into some advanced structures that are the input for the rest of the system, as well as, for the applications running on top of it. The third part consists in hashing algorithms that correlate the asynchronous events for each active call. Finally, a fourth part of the system is responsible for rebuilding the sequences of the acquired events. This

part is based on an LR(0) parsing algorithm with a user provided grammar defining the rules followed by the radio interface procedures.

We have verified the performance of our platform by means of a theoretical analysis and by performing several experiments based on some real radio activity data. During our experimentation we have observed that the mean execution time per event remains practically independent of the number of events currently being processed by the system and of their arrival rates. On the other hand the execution time is about one tenth of the available time. These observations show that our system may be used as a base for some "near" real time analysis applications.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

AST	Abstract Syntax Tree
ADL	Acquisition description language
EIA	Electronics Industry Association
DDL	Data description language
HLR	Home register locator
IS	Interim standard
LL(k)	Left to right leftmost derivations with a look ahead of k symbols
LR(k)	Left to right rightmost derivations in reverse with a look ahead of k symbols
MSC	Mobile switching centre
PCS	Personal communication system
TDMA	Time division multiple access
TIA	Telecommunication industry association
UCT	Unité centrale de traitement

INTRODUCTION

Définition du problème

La version nord américaine du système de téléphonie mobile de type TDMA («Time Division Multiple Access») est décrite par la norme ANSI-136 [16]. Cette norme présente, entre autres, l'interface radio et plus particulièrement les ensembles de messages et de procédures qui permettent aux terminaux mobiles d'établir les communications téléphoniques. La compagnie Ericsson propose une implantation de la norme ANSI-136 dans sa gamme de commutateurs numériques à usage général AXE-10.

Les commutateurs numériques AXE-10 contiennent un logiciel de contrôle qui est composé de plusieurs sous-systèmes responsables de l'implantation des différentes parties de la norme ANSI-136. Ces sous-systèmes sont composés, à leur tour, des modules appelés «blocs» qui communiquent entre eux par le biais de messages asynchrones. Chacun des blocs est responsable d'un ou plusieurs états de l'appel téléphonique. C'est pourquoi les messages envoyés entre les blocs membres du sous-système régissant l'interface radio reflètent fidèlement son activité. D'ailleurs, la plupart de ces messages correspondent directement aux événements qui surviennent entre les terminaux mobiles et le commutateur numérique, tels que décrits par la norme ANSI-136. Nous pouvons représenter un commutateur numérique par un système de contrôle par événements discrets qui peut être modélisé par un générateur de langage formel. Les symboles terminaux de ce langage sont constitués par les événements asynchrones définis par l'implantation de la norme ANSI-136 au sein de l'AXE-10. La grammaire de ce langage est décrite par les différentes procédures et machines à états spécifiées dans cette norme.

Dans ce projet, nous nous proposons de concevoir et de bâtir le prototype d'une plate-forme d'acquisition des événements asynchrones. Cette plate-forme servira de fondation pour des applications d'analyse évoluées fonctionnant en temps «presque» réel. Pour des raisons pratiques que nous expliquerons plus tard, la plate-forme d'acquisition doit se trouver à l'extérieur du commutateur numérique. Du point de vue fonctionnel la plate-forme doit effectuer quarte étapes de traitement sur les données avant que celles-ci ne soient livrées à l'application d'analyse. La première étape consiste à extraire les données du commutateur et à les transférer vers la plate-forme extérieure où le reste du travail sera effectué. La deuxième étape consiste à transformer ces données en un format évolué pour la plate-forme d'acquisition qui sera utilisée lors des étapes subséquentes, ainsi que par l'application d'analyse. La troisième étape consiste à trier les événements asynchrones afin de séparer les événements relatifs aux différents appels. Finalement, la troisième étape, qui constitue le cœur de la recherche, consiste à reconstruire les séquences d'événements acquis. Lors de cette étape nous allons utiliser les algorithmes d'analyse syntaxique afin de reconstruire ces séquences.

Méthodologie

Actuellement, il existe de nombreux outils d'analyse au sein de l'AXE-10. Notre système doit corriger les désavantages dont souffrent les outils actuels. C'est pourquoi nous allons dans un premier temps établir une liste de contraintes que notre système devra respecter afin d'offrir une solution supérieure aux solutions existantes.

Ensuite, en vertu de nos objectifs et des contraintes, nous allons dresser un aperçu systémique de notre plate-forme et nous allons décrire l'architecture de celle-ci. Afin de valider notre conception architecturale nous allons effectuer une implantation des fonctionnalités de base afin de pouvoir réaliser une expérience préliminaire qui servira de preuve de concept pour la continuation de notre projet.

Puisque, le cœur de notre projet est constitué d'algorithmes d'analyse syntaxique, nous allons nous pencher, par la suite, sur deux problèmes essentiels. Le premier problème consiste à déterminer le niveau de langage qui permettra de représenter et décrire les spécifications de la norme ANSI-136. Le deuxième problème consiste à choisir une méthode d'analyse syntaxique qui permet d'analyser les séquences d'événements décrites avec notre langage. Pendant cette étape nous généraliserons aussi notre approche en définissant deux langages. Le langage DDL («Data Description Language»), qui permettra de décrire les structures des événements reçus de l'AXE-10, ainsi que le langage ADL («Acquisition Description Language»), qui permettra de décrire la grammaire qui représentera le fonctionnement de l'implantation de la norme ANSI-136 au sein de l'AXE-10.

Finalement, nous effectuerons une brève évaluation des performances de tous les algorithmes choisis durant le projet. Ce sommaire sera suivi par une série d'expériences sur des données réelles qui permettront de valider notre méthode.

Résultats anticipés

Tel que nous l'avons précisé plutôt, l'objectif ultime de notre système est de servir de fondation aux applications d'analyse élaborées. Ces applications doivent pouvoir produire des résultats en temps «presque» réel. C'est pourquoi tout au long de notre projet nous devrons choisir des méthodes dont l'exécution est constante en temps et espace pour le traitement des événements individuels. Si nous réussissons dans cette tâche et que notre système est en mesure de traiter les événements sans variation significative de délai, il nous sera possible d'atteindre notre objectif. Cependant, il y a certains paramètres que nous ne connaissons pas assez bien ou qui sont difficiles à prédire. Il s'agit là des distributions des taux d'arrivée des événements pendant une période de 24 heures et de

l'efficacité de la plate-forme physique (processeur, et système d'exploitation) que nous allons choisir. C'est pourquoi il nous est très difficile de prévoir les résultats finaux.

Contenu du mémoire

Le mémoire est divisé en cinq chapitres principaux. Le premier chapitre donne un bref aperçu de la littérature qui a servi de base et de source d'informations pour ce projet. Dans le deuxième chapitre, nous allons justifier la réalisation de notre projet en présentant les contraintes auxquelles le système devra se plier afin d'apporter une amélioration aux méthodes d'analyse qui existent actuellement. Dans le troisième chapitre, nous allons présenter l'architecture complète de notre solution. Nous allons y présenter aussi l'expérience préliminaire avec ses résultats. Dans le quatrième chapitre nous allons discuter des problèmes principaux de notre travail c'est-à-dire des classes de langages, de la méthode utilisée lors de la reconstruction des séquences, ainsi que des structures des deux langages conçus lors de ce projet. Enfin, dans le cinquième chapitre nous allons présenter l'étude théorique des performances ainsi que les résultats expérimentaux obtenus avec les données réelles. Nous allons conclure en faisant un sommaire des résultats obtenus et en donnant des indications pour les travaux à venir.

CHAPITRE I - REVUE DE LA LITTÉRATURE

Dans le présent projet il s'agit de concevoir une plate-forme d'acquisition d'événements asynchrones. Ces événements proviennent de l'interface radio d'un commutateur numérique utilisé en téléphonie mobile. La plate-forme d'acquisition doit permettre la reconstruction de séquences d'événements et leur agrégation en données statistiques. Elle doit aussi permettre l'exécution d'actions associées à chacune des séquences définies par les usagers. Pour ce faire, des algorithmes d'analyse syntaxique sont utilisés. La définition des séquences est réalisée à l'aide d'une grammaire sans contexte basée sur la connaissance de l'implantation des normes régissant l'interface radio. Le travail de recherche est centré sur les techniques d'analyse syntaxique. Cependant, afin de pouvoir comprendre et résoudre le problème, il nous faut étudier plusieurs autres domaines de la recherche. Pour cette raison nous proposons une revue de littérature qui aborde le problème de plusieurs points de vue différents et complémentaires.

1.1 Source des événements asynchrones et situation actuelle

Un des premiers éléments à analyser est la source des événements asynchrones au sein du commutateur numérique. Dans [13] les auteurs[1] décrivent l'architecture du commutateur numérique AXE-10 d'Ericsson utilisé dans la présente recherche. Les

---

[1] Plusieurs des documents de formation ou décrivant les différents produits de Ericsson inc. ne présentent pas les noms des auteurs. Pour cette raison les références sans nom reviennent à plusieurs reprises dans ce mémoire.

auteurs mettent l'accent sur l'architecture logicielle de haut niveau. Notamment, ils présentent la structure des sous-systèmes et des blocs d'exécution (les programmes). Ils présentent aussi les différents types de messages permettant la communication entre les blocs d'exécution, en décrivant leurs formats et les avantages de chacun des types de messages. En particulier, les auteurs précisent que les messages du type «combiné» sont les plus efficaces. C'est pourquoi, nous utiliserons les messages de ce type afin de recueillir les informations des différents blocs du sous-système radio.

Nguyen et Ragosta [14] décrivent l'implantation de la norme ANSI-136 [16] au sein du commutateur numérique. En effet, ce document couvre toutes les phases d'un appel téléphonique utilisant plusieurs sous-systèmes. Dans ce projet nous nous intéressons uniquement à l'interface radio. D'autre part, dans [17] les auteurs décrivent toutes les règles de conception de l'interface radio selon la norme ANSI-136 [16]. Il s'agit surtout d'un document décrivant la mise en œuvre du système en utilisant les produits et les solutions conçues par Ericsson. Un certain nombre de statistiques conventionnelles fournies par le commutateur y est décrit. Dans le cadre de ce projet nous concevons un système d'analyse beaucoup plus flexible et permettant d'obtenir des statistiques plus évoluées que les systèmes classiques décrits dans [17]. Kelly Coursey [36] présente de manière générale tous les aspects de la norme ANSI-136. Il présente aussi son implantation, son intégration et son évolution dans les systèmes typiques.

Minichiello [39] décrit des expériences effectuées avec l'approche préconisée dans le présent projet mais avec des systèmes bâtis exclusivement pour ces expériences. Dans notre projet, nous concevrons un système générique permettant de construire rapidement des moteurs d'acquisition permettant d'effectuer ce genre d'analyses.

1.2 Systèmes de contrôle par événements discrets

Les systèmes dynamiques à événements discrets peuvent être modélisés par des automates dont l'évolution dynamique est déterminée par les séquences d'événements

discrets [21]. Ce modèle est souvent appelé «générateur de langage formel» [22]. Un commutateur numérique gérant la téléphonie mobile ou un réseau de téléphonie commuté conventionnel peut être considéré comme un système de contrôle à événements discrets [23]. Thistle et al [23] présentent une méthode de modélisation de contrôleurs pour des systèmes fonctionnant par événements discrets. Les exemples présentés concernent un réseau téléphonique simplifié. Thistle et al expliquent que ce système est modélisé comme dans [22] par un générateur de langage formel, dont l'ensemble des événements constitue l'alphabet du langage en question. Les «phrases» représentent les séquences des événements pouvant survenir. Le rôle du contrôleur est de désactiver de manière dynamique certains événements contrôlables afin d'obtenir le comportement désiré. Les spécifications des contrôleurs de ces systèmes prennent la forme d'un langage, le système contrôlé ne pouvant générer que des «phrases valides». Dans notre projet, il ne s'agit que d'observer les séquences d'événements discrets pouvant survenir dans l'interface radio d'un commutateur régissant les communications sans fil. Un des nos objectifs est de déterminer dans quelle mesure il est possible d'analyser les séquences de ces événements à l'aide d'un langage qui respecte les contraintes LR(0), ou plutôt, quel sera le comportement d'un tel système dans un environnement réel.

1.3 Systèmes d'acquisition et de traitement de données en temps réel

Yamashita et al [1] traitent d'un système d'acquisition et d'analyse en temps réel. Il s'agit de données de mesure générées par un accélérateur de particules. L'architecture du système est basée sur plusieurs postes de travail UNIX interconnectés par un réseau TCP/IP utilisant l'Ethernet 10Base-T (norme IEEE 802.3j-1993) comme couche physique. Ils proposent un traitement de données sous forme de pipeline, c'est-à-dire, que les données sont passées d'une poste de travail à l'autre pour effectuer les différentes transformations. Cette approche rend le système flexible et sa capacité facilement adaptable aux variations des contraintes. D'autre part, ils proposent de partager le réseau en deux parties : une pour les transferts de données à partir de la source vers les postes

effectuant le traitement et l'autre pour les accès des usagers. Chacun de ces postes, en effet, possède deux interfaces TCP/IP. De cette manière, le trafic des données est isolé de l'activité du réseau d'accès.

Les données acquises par le système conçu sont reçues sous forme de blocs. Chacun de ces blocs contient un nombre variable de structures de données présentant les événements asynchrones. Belovitch et al [2] décrivent un système de stockage flexible. Ce système permet de stocker des séries d'échantillons acquis à des fréquences variables. Toutes les données acquises sont groupées en blocs et les recherches se font de façon dichotomique de manière à obtenir une efficacité de $\log_2(n)$ où n est le nombre de blocs stockés. Notons aussi que le système de stockage présenté peut être accédé en lecture par plusieurs usagers simultanément pendant que le système est en acquisition.

Arras et al [53] présentent la notion de temps «presque» réel («soft-real-time») comme caractéristique de certains systèmes de communication par paquets qui permettent des délais variables et même des pertes de données. Il s'agit surtout de systèmes de transferts de données sonores ou vidéo. Le système que nous voulons concevoir doit permettre un stockage et une analyse continue sur une plate-forme peu coûteuse et flexible. Nous voulons nous limiter à offrir les résultats dans des délais raisonnables sans offrir la possibilité d'implanter une boucle de contrôle [54]. C'est pourquoi, notre système doit se comporter comme un système fonctionnant en temps «presque» réel, sans pertes de données.

Le système conçu est un système d'observation d'un réseau téléphonique sans fil. Le débit des événements acquis dépend directement du trafic téléphonique. Bratt [11] présente des exemples de distribution du débit de trafic téléphonique ventilé selon les heures de la journée, les jours de la semaine et les mois de l'année. Il présente aussi le modèle établi par le chercheur danois Erlang ainsi que les règles dites «heures de pointe» utilisées lors du dimensionnement du réseau téléphonique. On voit qu'il est impossible de décrire le comportement du réseau à long terme par une distribution statistique simple. Les

arrivées des appels suivent la loi de Poisson mais le taux d'arrivées est variable. Plus particulièrement, il dépend des activités des usagers, des plans de service offerts, des jours de la semaine, des mois de l'année, etc. Le débit des données acquises par le présent système va suivre exactement le même comportement, puisque l'activité de l'interface radio dépend directement du comportement des usagers.

1.4 Systèmes de traitement de données avancées «Data Mining»

Le «Data Mining» présentée par Chen et al [4] et Colin [5] permet de stocker d'abord les données dans des bases de données spécialisées [5] et ensuite de rétablir les relations en faisant des jointures entre les différentes tables d'événements de manière efficace. Cette technique est souvent proposée dans les situations où on ne connaît pas la relation entre les différentes données. C'est pourquoi plusieurs chercheurs tels que Nakajima [6], Hirota, Pedrycz [7, 8], Baldwin [9] et Lin [10] étendent cette technique à l'aide de la logique floue qui permet d'élargir la notion d'ensembles afin d'accroître le nombre de possibilités. Selon Askerup [47], dans notre cas, le «Data Mining» permettrait d'effectuer des analyses beaucoup plus poussées. Puisque l'acquisition et l'organisation initiale des données sont essentielles [5, 47], notre plate-forme doit permettre d'organiser les données, de manière à ce que des techniques telles que la généralisation [4] soient plus faciles à réaliser.

1.5 Applications similaires des algorithmes d'analyse syntaxique

L'aspect essentiel du présent système est la reconstruction des séquences d'événements asynchrones en utilisant des algorithmes d'analyse syntaxique. Yost et al [12] décrivent une méthode permettant d'extraire les informations dites «sémantiquement intéressantes» à partir des résultats de simulation. Ils décrivent aussi un langage de définition des événements dits «significatifs». En effet, ce langage sert à décrire une machine à états qui inspecte les données lors de la simulation et en construit un arbre

d'analyse syntaxique («parse tree»). Si certaines séquences d'événements sont détectées et que les données qui contribuent à ces séquences ont des valeurs respectant des règles sémantiques définies par l'usager, les résultats sont convertis en représentation graphique.

Bobick et Ivanov [19] décrivent un système qui permet de reconnaître les gestes de la main. Ce système est conçu en deux couches : la première détecte les primitives du mouvement et la deuxième détecte les gestes en utilisant l'analyse syntaxique probabiliste. L'analyse syntaxique probabiliste est basée sur les algorithmes généraux d'analyse syntaxique des grammaires sans contexte de Earley étendue par Stolcke [20] pour les grammaires probabilistes. Dans cette méthode tout comme dans le présent projet les auteurs basent le processus de détection des gestes sur la connaissance à priori des séquences de mouvements pouvant survenir. Dans ce cas particulier, l'univers des possibilités est considérable car il y a très peu de règles fiables qui régissent cet univers. C'est pourquoi, ils doivent utiliser un algorithme général. Dans le présent projet le nombre de possibilités est limité et bien défini par les normes et par l'implantation logicielle de ces normes. C'est pourquoi nous avons choisi d'utiliser un algorithme LR(0) beaucoup moins puissant, mais plus efficace en espace et en temps.

1.6 Littérature générale sur les algorithmes d'analyse syntaxique

Le cœur de notre travail de recherche est constitué par différentes méthodes d'analyse syntaxique. Il s'agit là aussi bien de la théorie que des méthodes d'implantation. Grune et Jacobs [27] décrivent le problème d'analyse syntaxique de façon très générale. Ils présentent les algorithmes pour les approches LL («Left to Right, Left Recursive») et LR («Left to Right, Right Recursive»). Ils traitent des algorithmes généraux, comme l'algorithme de Earley [27], qui constituent plutôt des méthodes théoriques. Ils présentent aussi les méthodes pratiques telles que la méthode LALR («Look Ahead LR»). Parsons [26] fait en quelque sorte une introduction au domaine de l'analyse syntaxique et à la construction des compilateurs en ne présentant que les méthodes de base. Aho, Sethi et

Ullman [24] présentent un véritable recueil de recettes décrivant un grand nombre de méthodes pratiques disponibles pour effectuer l'analyse syntaxique ainsi que la génération automatique des programmes d'analyse. Finalement, Chapman [25] décrit l'analyse syntaxique de manière très formelle, en faisant des preuves très rigoureuses pour chaque élément de l'analyse syntaxique.

1.7 Littérature générale sur les algorithmes de base et le design

Un autre des principaux éléments de notre projet est l'implantation de l'algorithme d'analyse syntaxique du type LR(0). Nous voulons vérifier jusqu'à quel point il est possible de reconstruire les séquences d'événements et dans quelle mesure cette approche permet d'obtenir des résultats en temps «presque» réel. Cette partie du projet requiert des connaissances approfondies dans le domaine des algorithmes afin que le logiciel conçu soit efficace. Knuth [28, 29] et Sedgewick [30] décrivent les différents algorithmes de base. La référence [30] est plutôt un document traitant des algorithmes de manière générale, tandis que les références [28, 29] constituent des discussions très détaillées décrivant l'efficacité des algorithmes du point de vue du temps et de l'espace d'exécution.

L'implantation comprend aussi la génération de matrices LR(0) à partir des spécifications décrites dans un langage conçu dans le cadre du présent projet. Aho, Sethi et Ullman [24] décrivent plusieurs méthodes pour générer les matrices LR(0). Lemone [34] donne une méthode de haut niveau de conception d'un compilateur LR(0). Sethi [35] décrit la forme Backus-Naur de description de grammaires sur laquelle le langage de description des acquisitions («Acquisition Description Language», ADL) de notre projet est basé. Levine, Mason et Brown [33] offrent un guide pratique des outils LEX et YACC utilisés dans le présent projet pour générer les compilateurs du langage permettant de décrire les données DDL («Data Description Language») et du ADL.

Enfin, toute l'implantation du présent projet est basée sur la méthode orientée objet et est programmée en C++ [31]. La plupart des concepts de design sont décrits par Gamma et al [32] qui présentent un véritable recueil de méthodes de design orienté objet.

CHAPITRE II – DÉFINITION DU CADRE DE RECHERCHE

Dans ce chapitre, nous allons présenter la problématique du système conçu dans le cadre de notre projet. Tel qu'énoncé dans les chapitres précédents, il s'agit d'une plate-forme flexible permettant l'acquisition d'événements asynchrones. Ces événements correspondent à la signalisation de l'interface radio d'un réseau sans fil qui suit la norme ANSI-136 [16]. Cette interface est gérée par un commutateur numérique AXE-10 d'Ericsson. L'objectif principal est de permettre aux ingénieurs d'analyser le comportement de l'interface radio par le biais de statistiques et d'observations en temps «presque» réel. Nous commencerons notre discussion par une brève description de la norme ANSI-136 et par un aperçu de l'architecture logicielle du commutateur. Ensuite, nous présenterons la situation actuelle, la motivation du projet, ainsi que les contraintes et objectifs. Nous continuerons avec la présentation systémique de la plate-forme d'acquisition. Enfin, nous conclurons par une description des limites de notre projet.

2.1 La norme ANSI-136

La norme ANSI-136 [16] a été mise en place par le groupe TR-45.3 de la TIA (Telecommunication Industry Association) qui travaille en association avec la EIA (Electronic Industry Association). Cette norme est une agrégation de plusieurs normes et protocoles du PCS (Personal Communication Service) numérique tels que IS-136, IS-137, IS-138, IS-641, IS-130 et IS-135 de la TIA («IS» voulant dire «Interim Standard ») [36]. Elle définit tous les aspects du PCS, des protocoles physiques jusqu'à la définition des

services. Dans le cadre de ce projet, nous sommes uniquement intéressés par le fonctionnement logique de l'interface radio. À ce sujet, la norme définit une série de machines à états et algorithmes qui décrivent les différentes procédures utilisées. Nous allons revenir sur ce point dans les chapitres ultérieurs. Un autre aspect important de la norme ANSI-136 pour notre travail de recherche est l'architecture typique du réseau [36] dont une version simplifiée est présentée à la figure 2.1. Sur cette figure nous remarquons que l'architecture comprend des stations d'antennes BS («Base Stations») reliées au centre de commutation mobile MSC («Mobile Switching Centre»). Le MSC gère l'interface radio aussi bien que les accès aux lignes du réseau commuté de téléphonie publique PSTN («Public Switched Telephony Network»), ou l'accès aux lignes des autres MSC. Cette architecture est une particularité de la norme ANSI-136 car la plupart des autres normes comme les normes GSM [37], CDMA IS-95 [38] ou cdma2000 [38] comprennent aussi un contrôleur de stations d'antennes BSC («Base Station Controller»), qui est exclusivement responsable de la gestion de l'interface radio. Dans ces cas, la responsabilité du MSC est réduite à la gestion des accès au PSTN et aux autres MSC. Dans notre projet la plate-forme d'acquisition devra interagir directement avec le MSC.

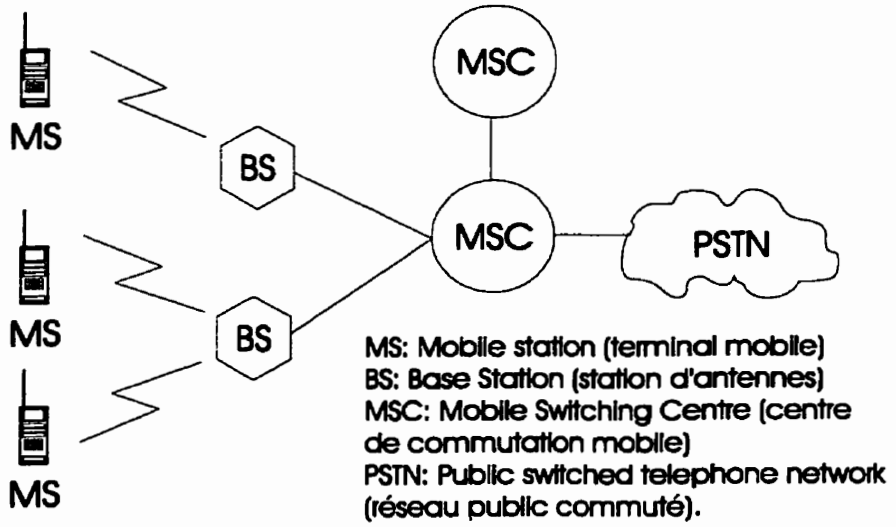

Figure 2.1 Architecture typique du réseau selon la norme ANSI-136

2.2 Architecture du commutateur AXE-10 d'Ericsson

Tel qu'il a été mentionné plus tôt, l'élément clé en communications mobiles régies par la norme ANSI-136 est le MSC ou dans notre cas l'AXE-10 d'Ericsson. Il remplit fonctions principales : premièrement, il régit l'interface radio, deuxièmement, il permet d'établir les communications entre les différents usagers du système téléphonique. Le commutateur numérique requiert un logiciel de contrôle écrit dans un langage spécial appelé PLEX («Programming Language for Exchanges») [13]. D'une part, ce logiciel est subdivisé en «blocs». D'autre part, un appel téléphonique peut être visualisé comme une machine à états. C'est pourquoi, chacun de ces blocs est responsable d'un ou plusieurs états de l'appel. Au fur et à mesure qu'un appel évolue (change d'état) les ressources sont saisies au sein des différents blocs. Cela forme une «chaîne d'appel» («call path» [14]). La figure 2.2 donne un exemple très simplifié de l'établissement d'un appel. Tout d'abord, un usager compose un numéro qui est décodé par le bloc «A». Ensuite, le bloc «B» vérifie le numéro demandé auprès d'un registre des numéros valides. Par la suite, le bloc «C» localise et vérifie la ligne de l'usager appelé. Finalement, un bloc «D» établit la communication entre les deux usagers. La communication entre les blocs est effectuée par l'envoi de signaux. Lorsqu'un bloc a fini de faire sa part de travail (il est alors temps de changer d'état) il envoie un signal vers le bloc suivant. Chaque signal est un message asynchrone qui contient les informations nécessaires (sur l'appel en cours) pour que le bloc suivant puisse exécuter sa tâche. Une fois à l'intérieur du bloc, les signaux sont copiés dans une structure appelée «enregistrement» («record») pour toute la durée de l'appel. Cela correspond à la saisie des ressources au sein des blocs et à la création de la «chaîne d'appel». Les signaux sont en effet des événements asynchrones. Les données contenues à l'intérieur de ces événements pourraient être observées afin de surveiller l'activité du commutateur numérique. Elles pourraient aussi être stockées afin d'effectuer des analyses plus élaborées sur le fonctionnement du réseau (e. g. «Data Mining»). C'est en effet ce que nous nous proposons de faire dans ce projet. Cependant, nous sommes

uniquement intéressés par l'activité de l'interface radio. Notre système ne permettra donc que d'acquérir les signaux envoyés par les blocs qui gèrent l'interface radio.

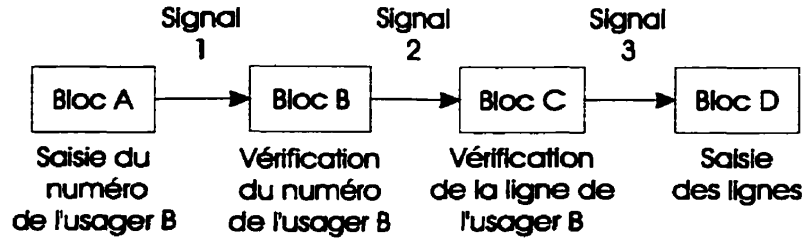

Figure 2.2 Exemple simplifié d'établissement d'un appel

2.3 Situation actuelle

Actuellement, au niveau du commutateur numérique, il existe de nombreux outils pour observer son fonctionnement. Tous ces outils souffrent de trois désavantages majeurs :

1. Il s'agit d'outils intégrés au commutateur dont le fonctionnement augmente la charge de l'UCT (Unité Centrale de Traitement) au point que de nombreuses sociétés exploitantes interdisent leur utilisation durant les heures de pointe. Or, c'est pendant ces heures, que les observations seraient les plus intéressantes.

2. Comme ces outils sont intégrés au logiciel de contrôle du MSC, leur développement doit suivre les règles strictes du développement de ce logiciel. Or, il est formellement interdit de faire des changements rapides et non certifiés car ceci pourrait avoir des répercussions négatives sur la fiabilité du système. C'est pourquoi il est impossible de les adapter ou de les optimiser pour des analyses particulières.

3. L'AXE-10 est une plate-forme entièrement «propriétaire», c'est-à-dire qu'elle se base uniquement sur des produits développés à l'interne. C'est pourquoi, elle offre très peu de possibilités d'interopérabilité avec d'autres systèmes commerciaux tels que les systèmes de fichiers réseaux, bases de données ou systèmes d'interfaces graphiques. Il est donc difficile, voire impossible, d'intégrer au logiciel de contrôle des méthodes et outils commerciaux qui permettraient d'effectuer des analyses plus poussées.

2.4 Motivation et idée générale

Étant donnés les inconvénients des outils actuels, nous nous proposons de concevoir une plate-forme d'acquisition des événements asynchrones (signaux) du logiciel de contrôle correspondant à l'activité de l'interface radio. Cette approche garantira un maximum de flexibilité aux ingénieurs et permettra d'effectuer des analyses complexes en temps «presque» réel lors des séances d'acquisition («online analysis»), aussi bien qu'après que les séances d'acquisition soient terminées («offline analysis»). En effet, cette méthode doit aller totalement à l'encontre des méthodes traditionnelles qui ont tendance à offrir du «tout cuit», mais avec un nombre limité de possibilités. Elle doit permettre de faire des traitements qui incluent du «data mining» avec les données obtenues, tout en minimisant le taux d'utilisation de l'UCT lors de son fonctionnement.

2.5 Contraintes et objectifs

La solution proposée doit respecter cinq contraintes principales pour que le système conçu apporte une amélioration à la situation actuelle ou plutôt, qu'il corrige les lacunes des méthodes actuelles.

1. **Traitement externe et minimisation de l'utilisation de l'UCT** : D'une part, il faut minimiser le taux d'utilisation de l'UCT. D'autre part, un commutateur ne possède pas d'unité de stockage dont la capacité pourrait être dédiée au stockage des données acquises. Ceci implique que le traitement et le stockage doivent se faire à l'extérieur du MSC.

2. **Corrélation et reconstruction des séquences** : Seuls, les événements acquis n'offrent que très peu d'informations. En effet, ils ne permettent pas d'offrir plus d'informations que les systèmes existants. Cependant, les séquences d'événements reconstituées entièrement, ou en partie, permettent d'observer très exactement le fonctionnement de l'interface radio. Cette reconstruction doit se faire en trois étapes. La première étape consiste à extraire les événements d'intérêt pour l'analyse. La deuxième étape, consiste à classer les éléments reçus selon une clé qui identifie les instances sur lesquelles l'analyse est effectuée. Dans la plupart des cas, il s'agit d'analyser des appels en cours dans le réseau. La clé est alors l'identification des terminaux mobiles. En pratique, il s'agit du numéro de téléphone de ces terminaux. Finalement, la troisième étape consiste à reconstruire chaque séquence d'événements selon des règles connues d'avance, qui correspondent directement à la description de la machine à états analysée.

3. **Facilité, flexibilité et ouverture pour d'autres applications** : Le système doit être facilement modifiable et aussi flexible que possible afin de maximiser les possibilités d'analyses. En effet, il doit offrir une interface de programmation permettant aux usagers de l'utiliser de manière facile sans connaître les détails de son implantation. Il doit aussi pouvoir être intégré avec n'importe quelle application disponible sur le marché.

4. **Traitement en temps «presque» réel** : Tel nous l'avons expliqué dans le chapitre précédent, le traitement doit s'effectuer en temps «presque» réel sans pertes de données, afin de permettre l'observation continue du commutateur. Ceci impose l'utilisation d'algorithmes dont l'exécution est linéaire en temps et en espace. Il est important de rappeler que dans le cadre du présent travail il ne s'agit pas d'offrir des bases pour bâtir une boucle de contrôle [54]. C'est pourquoi nous considérons uniquement le traitement en temps «presque» réel avec des délais de dépassant pas plus que 5 secondes.

5. **Infrastructure peu coûteuse** : La plate-forme physique ainsi que l'interface avec le MSC doivent être le plus communes possibles, peu coûteux, facilement transportables et faciles à installer. Ceci doit permettre aux ingénieurs d'effectuer les analyses sans investir au préalable beaucoup de temps, d'argent et d'énergie pour mettre en place toute l'infrastructure.

2.6 Aperçu systémique de la plate-forme

En vertu des contraintes énumérées dans la section précédente nous pouvons établir une image systémique de la plate-forme. Cette image est présentée à la figure 2.3. On peut y distinguer les caractéristiques suivantes. Tout d'abord, la plate-forme d'acquisition fonctionne à l'extérieur du MSC (contrainte 1, «Traitement externe et minimisation de l'utilisation de l'UCT»). Il s'agit, là, de n'importe quel type d'ordinateur personnel (ou poste de travail) relié au MSC via une interface TCP/IP, ce qui permet d'utiliser la plupart des outils disponibles sur le marché (contrainte 3, «Flexibilité et ouverture pour d'autres applications»). Ceci rend l'équipement peu coûteux, facile à installer et surtout facilement accessible (contrainte 5, «Infrastructure peu coûteuse»). Ensuite, la plate-forme est sous-divisée en deux parties principales. La première partie est caractérisée par les algorithmes constants, indépendants du type d'analyse exécutée. La

deuxième partie constitue l'implantation de l'analyse effectuée. Entre ces deux parties se trouve l'interface de programmation (contrainte 3, «Flexibilité et ouverture pour d'autres applications») disponible aux usagers.

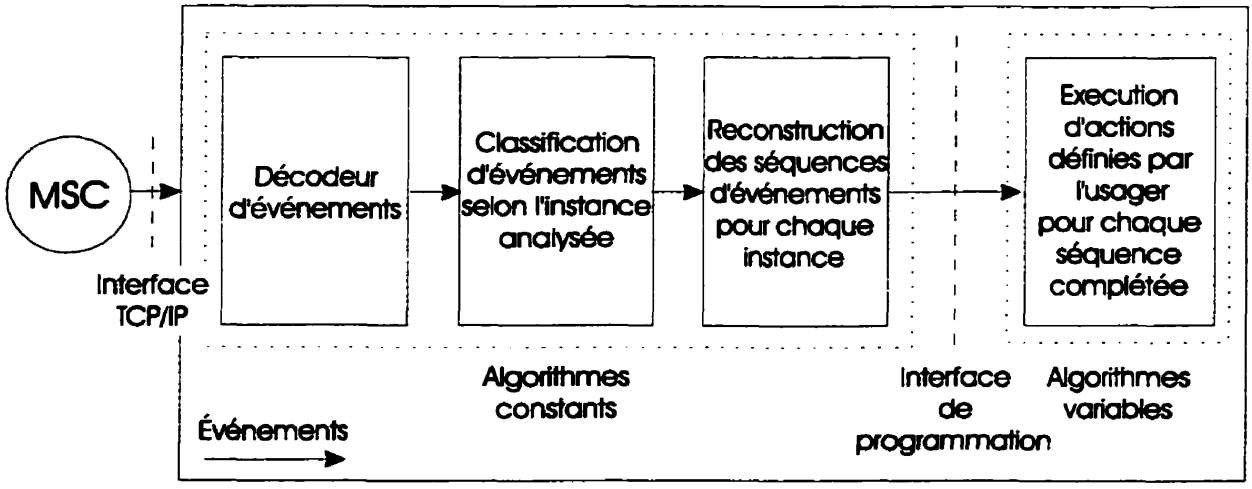

Figure 2.3 Aperçu systémique de la plate-forme d'acquisition

Le flux des données est présenté à la figure 2.3 où nous pouvons remarquer quatre étapes spécifiques. Premièrement, à l'entrée du système les événements doivent être traduits en un format utile pour le reste du traitement. Dans les chapitres ultérieurs, nous allons voir que ces événements arrivent en groupes pour des raisons d'efficacité. Cette étape permet aussi d'extraire les événements qui sont intéressants pour l'analyse subséquente (contrainte 2, «Corrélation et reconstruction des séquences»). Deuxièmement, tous les événements doivent être classés selon l'instance qui est analysée. Comme il a été mentionné plus tôt (contrainte 2, «Corrélation et reconstruction des séquences») l'analyse consiste à observer plusieurs instances actives simultanément, donc les événements doivent être classés selon une clé identifiant ces instances. Troisièmement, l'activité de chaque instance est décrite par une série de machines à états décrivant les différentes procédures (contrainte 2, «Corrélation et reconstruction des séquences»). La responsabilité de cette partie est de déclencher des actions spécifiques chaque fois qu'une procédure ou

une partie de cette procédure est exécutée. Plus tard nous verrons comment l'utilisation des algorithmes d'analyse grammaticale du type LR(0) permettra une exécution en temps «presque» réel (contrainte 4, «Traitement en temps «presque» réel»). Finalement, la quatrième partie est constituée par un ensemble d'actions particulières implantées pour chaque type d'analyse (contrainte 3, «Flexibilité et ouverture pour d'autres applications»).

2.7 Limites du cadre de recherche

Tel que présenté à la figure 2.3, dans le cadre de ce projet allons nous préoccuper de la partie générique du flux de données se trouvant l'intérieur de la plate-forme d'acquisition. L'interface de programmation constitue la limite de nos responsabilités. Plus particulièrement, le cœur du travail sera constitué par l'analyse et l'implantation des algorithmes de décodage, de classification et de reconstruction des séquences d'événements. Ces événements vont nous parvenir des blocs du MSC gérant l'interface radio. Sur un second plan, nous allons d'évaluer de façon générale comment la plate-forme ainsi conçue se comporte avec les données provenant d'un réseau réel.

CHAPITRE III – ARCHITECTURE DU SYSTÈME

Dans ce chapitre nous présenterons les détails de l'architecture du système en étendant la présentation systémique du chapitre précédent. Plus particulièrement, nous parlerons de l'implantation au sein du MSC. Ensuite, nous montrerons le détail de la plate-forme d'acquisition et nous allons terminer avec une présentation de haut niveau du traitement des données.

3.1 Implantation au sein du MSC

La première phase du processus d'acquisition se trouve au niveau du MSC, mais comme il a été mentionné dans le chapitre 2, la performance du MSC doit rester pratiquement inchangée. C'est pourquoi cette implantation doit rester minimale. Nous avons pris la décision d'implanter un bloc simple permettant uniquement d'extraire les événements du MSC. Ce bloc reçoit les copies de messages, qui sont passées entre les blocs du logiciel de contrôle, par le biais des messages dits «combinés» [13]. Le contenu de ces messages est copié dans un espace tampon qui est transféré vers la plate-forme extérieure à travers une interface TCP/IP [40, 43]. Ce transfert est effectué quand l'espace tampon est suffisamment rempli, ou après qu'un temps prédéfini se soit écoulé. Les messages dits «combinés» sont très efficaces car leur transfert est effectué en appelant une routine au sein du bloc cible sans passer par un stockage intermédiaire tel qu'une pile ou une file d'attente de messages [13]. Il s'agit là de ce que nous pourrions appeler un «goto»

avec paramètres. Il faut aussi noter que grâce aux procédures de maintenance[1], les messages combinés peuvent être facilement ajoutés quand le MSC est actif. Ceci rend notre approche particulièrement intéressante, car de cette manière, outre l'activité de l'interface radio, il sera possible d'observer le comportement de pratiquement n'importe quel bloc du logiciel de contrôle. La figure 3.1 schématise ce processus. On y voit que tous les blocs envoient une copie des messages via le message combiné avant de les envoyer vers leur destination finale.

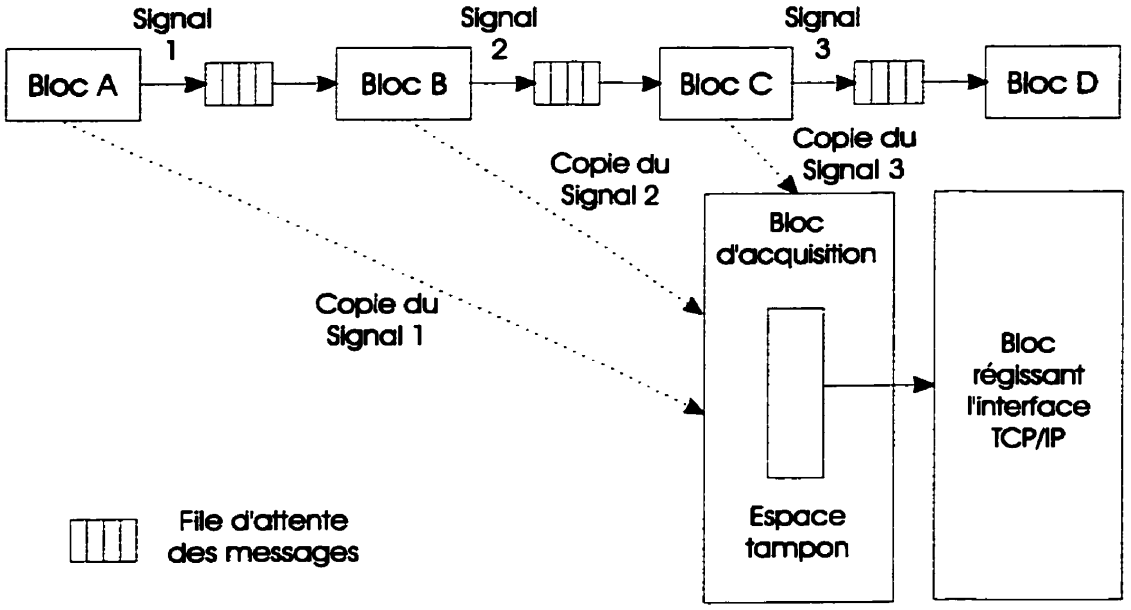

Figure 3.1 Block d'acquisition au sein du MSC

3.2 Plate-forme d'acquisition

Pour des raisons pratiques nous avons choisi un ordinateur personnel (poste de travail) avec le système d'exploitation Linux comme plate-forme extérieure physique.

---

[1] L'AXE-10, grâce à son architecture redondante, permet de changer des parties du code pendant qu'il est en utilisation.

Pour des raisons de flexibilité nous avons décidé d'utiliser une interface TCP/IP appelée «Signalling Terminal for Open Communications» (STOC) [40] pour la connexion entre le MSC et le poste de travail. Cette approche est inspirée d'un système conçu pour l'acquisition en temps réel des événements provenant d'un accélérateur de particules [1]. Les auteurs de ce système d'acquisition suggèrent que l'ordinateur, qui effectue les acquisitions, possède deux interfaces réseau, une qui sert exclusivement au lien avec la ou les sources de données et l'autre pour accéder à la plate-forme à partir d'un réseau corporatif. Nous avons suivi exactement la même approche.

3.2.1 Performances de l'interface

Dans notre cas le lien physique entre le MSC et le poste de travail est un lien Ethernet 10Base-T bidirectionnel («full duplex») suivant la norme IEEE 802.3i-1990. Ethernet offre un accès non déterministe au médium partagé, c'est-à-dire que n'importe quel élément du réseau peut commencer à transmettre à n'importe quel moment. Si deux éléments transmettent simultanément, une collision a lieu et les deux éléments doivent attendre un temps aléatoire avant de réessayer une transmission. Tel que nous avons expliqué dans la section précédente, nous avons décidé de connecter le MSC au poste de travail avec un lien isolé de tout autre trafic. Dans le présent projet, il s'agit d'un lien bidirectionnel sans autres applications. Nous pouvons donc négliger les pertes de performances dues aux collisions. L'Ethernet dont nous parlons offre une capacité théorique maximale 812 trames de 1500 octets par seconde (ou 9,744,000 bits/s) [42]. Dans notre cas il faut aussi comptabiliser 20 octets par trame pour l'entête des paquets IP et 20 octets pour l'entête TCP[1] ce qui réduit la capacité théorique maximale à 1,185,520

---

[1] Le protocole TCP, pour éviter la fragmentation, insère les paquets complets dans chaque paquet IP [43], donc nous pouvons assumer que tous les paquets envoyés contiennent un en-tête TCP.

octets/s (ou 9,484,160 bits/s). Bien évidemment, ces chiffres doivent être interprétés comme des ordres de grandeurs uniquement car l'efficacité exacte de l'interface dépend d'un grand nombre de paramètres, tels que l'efficacité des logiciels pilotes, l'efficacité de l'implantation du protocole TCP/IP, etc. En effet, la seule façon d'établir les performances réelles est d'effectuer un certain nombre d'expériences [42]. Nous n'avons pas pu trouver de rapports faisant part de telles expériences et, faute de temps, nous ne pouvions pas les faire nous-mêmes. C'est pourquoi, nous nous en tiendrons à des ordres de grandeur uniquement.

La capacité théorique du lien dépend de la taille des trames, et la taille maximale (utile) de celles-ci est de 1,500 octets [42]. En règle générale, plus les trames sont longues plus l'efficacité est élevée [41], c'est pourquoi nous avons décidé d'avoir une taille d'environ 8.5k octets pour l'espace tampon au sein du MSC. De cette manière, nous nous assurons de transmettre environ 6 trames de 1500 octets pour chaque espace tampon transmis. Il est important de remarquer que TCP est un protocole de transfert dit «fiable» du type GO BACK N [44] (ou à fenêtre coulissante) ce qui offre, entre autres, un mécanisme de contrôle de flux de données («data flow control») qui bloque le transfert si l'application recevant les données est saturée [43]. Dans ce cas, l'application qui envoie les données doit s'arrêter sinon des données seront perdues. Dans le présent projet, puisqu'il est impossible d'arrêter ou diminuer le taux d'arrivée des données à la source, car celui-ci dépend de l'activité de l'interface radio, il nous faut implanter une file d'attente où les données pourront être stockées. Or, comme nous voulons réduire l'impact sur le fonctionnement du MSC, nous avons décidé d'implanter cette file d'attente au niveau du poste de travail. Afin de réaliser ceci, nous avons divisé l'application au niveau du poste de travail en deux parties. La première partie est très simple, elle ne fait que recevoir les données et les stocker dans des fichiers sur le disque, la deuxième partie lit ces fichiers et effectue tout le traitement. De cette manière, le mécanisme de contrôle de flux de données ne peut pas être enclenché et provoquer des pertes de données. Ceci nous

permet aussi d'offrir la possibilité de conserver les données afin de pouvoir exécuter d'autres analyses sur des données déjà acquises («offline analysis»). La présence d'une file d'attente est d'autant plus importante que l'activité du MSC n'est pas constante, car elle dépend de l'heure de la journée, du jour de la semaine ou de la période de l'année [11]. Le système doit donc être en mesure de résorber des fluctuations de l'activité du MSC. D'autre part, avec cette approche, si le traitement est plus lent que les arrivées et que le lien TCP/IP n'est pas saturé, seul le délai va augmenter. Le système continuera à fonctionner normalement en consommant presque la totalité du temps de l'UCT mais il n'y aura pas d'effondrement. Finalement, cette architecture protège le système contre les pertes de données car en effet il y a deux applications : une très simple qui s'assure de recevoir des données et l'autre beaucoup plus complexe qui effectue le traitement sur les données acquises. De toute évidence, une application complexe est beaucoup plus susceptible d'échouer. Donc en gardant l'application qui assure le transfert de données simple et fiable, nous avons une probabilité plus faible d'avoir des pertes de données dues à des erreurs dans l'application. La figure 3.2 montre le système décrit dans cette partie. Ajoutons que le poste de travail va agir comme client TCP tandis que le MSC va agir comme serveur.

3.2.2 Expérience préliminaire

Il est très difficile de prévoir exactement quelle sera la capacité du MSC requise pour faire fonctionner le système correctement. Il est aussi difficile de prévoir quel sera le débit des données et comment il va s'approcher des caractéristiques maximales du lien physique. A l'heure actuelle, nous ne connaissons aucun ensemble de statistiques permettant de nous fournir cette information. Finalement, il faut aussi estimer quelle sera la charge au niveau du poste de travail. C'est pourquoi, afin de valider notre design de haut niveau nous avons effectué une expérience préliminaire.

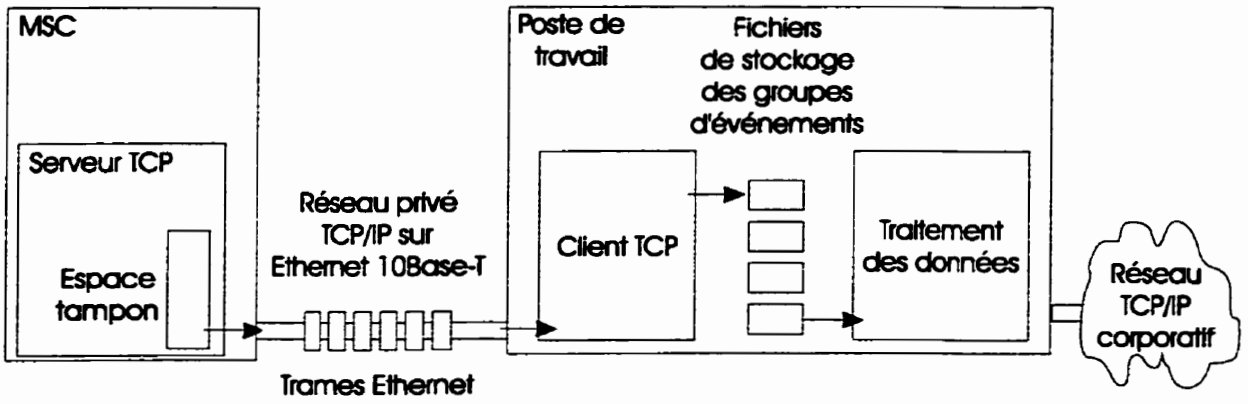

Figure 3.2 Architecture globale du système

Le système qui a été utilisé lors de cette expérience correspond exactement au système présenté à la figure 3.2. Le poste de travail était un ordinateur portatif de marque Compaq avec une UCT Pentium à vitesse d'horloge de 100MHz. La charge du poste de travail était observée pendant les heures de pointe à l'aide du programme utilitaire «top» de Linux. Le traitement consistait à décoder tous les 46 types d'événements et analyser en particulier les événements du type «Radio Quality Message» (ou RQM)[1] et du type «Originating Access» (ou OA)[2] afin d'afficher le contenu de ceux qui provenaient d'un ensemble de dix numéros de terminaux d'essai. Chaque fois qu'un groupe d'événement était reçu sa taille exacte et son temps d'arrivée étaient enregistrés. Le bloc d'acquisition au sein de l'MSC comptait le nombre d'occurrences des pertes de données résultant du mécanisme de contrôle de flux du TCP. Le commutateur était un commutateur de région urbaine, l'acquisition a duré pendant 25 heures (de 9h30 jusqu'à 10h30 le lendemain), les observations suivantes ont été réalisées :

---

[1] Les messages RQM sont envoyés par des terminaux mobiles en mode numérique à toutes les deux secondes afin de faire part des conditions de réception radio.

[2] Les messages OA sont envoyés par des terminaux mobiles à chaque fois qu'une connexion est requise par un terminal mobile.

1. Le taux d'arrivées des groupes d'événements est représenté à la figure 3.3 où nous pouvons remarquer que le taux maximal est de 36579 octets/s ce qui est environ trente deux fois inférieur à la capacité théorique maximale du lien physique. Nous pouvons donc considérer que le lien physique est suffisant pour les transferts de données lors des acquisitions.

2. La courbe suit les fluctuations décrites dans [11] lors des différentes périodes de la journée. D'ailleurs, la société exploitante chez qui l'acquisition a été effectuée offrait différentes périodes tarifaires, ce qui se traduit par des crêtes à 18h00, 20h00 et 21h00.

3. La charge du commutateur lors de l'acquisition durant des heures de pointe était supérieure de 3% par rapport à la charge habituelle durant ces mêmes heures.

4. La charge du poste de travail n'a pas dépassé 8% durant les heures de pointe et s'est maintenue à environ 4% en dehors de ces heures. L'application qui recevait les données ne générait pas plus de charge qu'environ 1%.

5. Le délai approximatif des messages OA entre la composition du numéro et leur apparition à l'écran était entre 3 et 4 secondes durant les heures de pointe. Notons, que ce délai reste très approximatif car nous ne disposions pas d'équipement permettant de mesurer cette valeur avec précision La figure 3.4 présente la distribution du taux d'arrivée des messages du type OA et la figure 3.5 présente la distribution du taux d'arrivée des messages RQM.

6. Durant toute la période d'acquisition il n'y a eu aucune occurrence de perte de données due au mécanisme de contrôle de flux du TCP. D'autre part, il n'y a pas eu de débordement de fichiers de stockage.

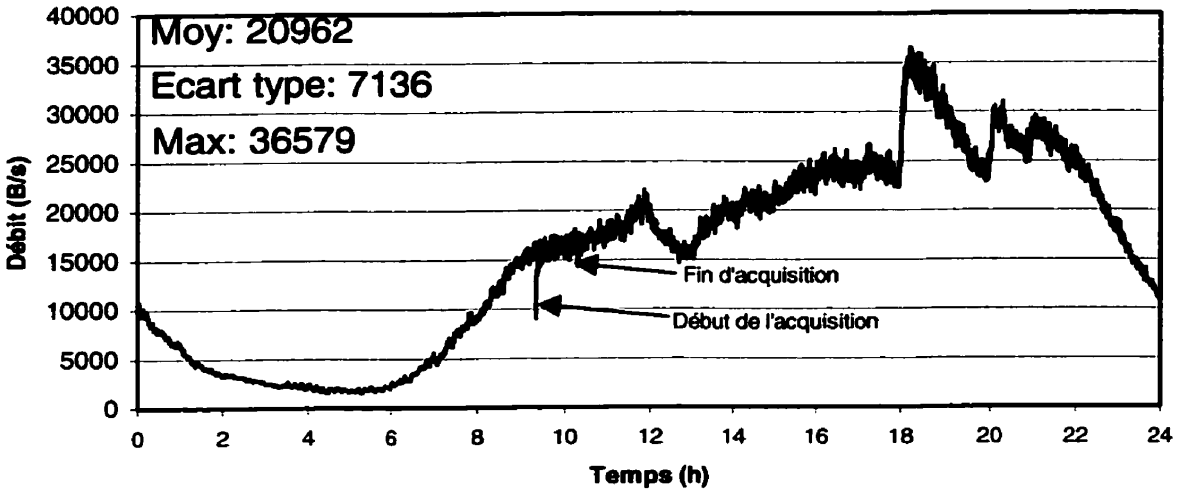

Figure 3.3 Distribution du débit total des données

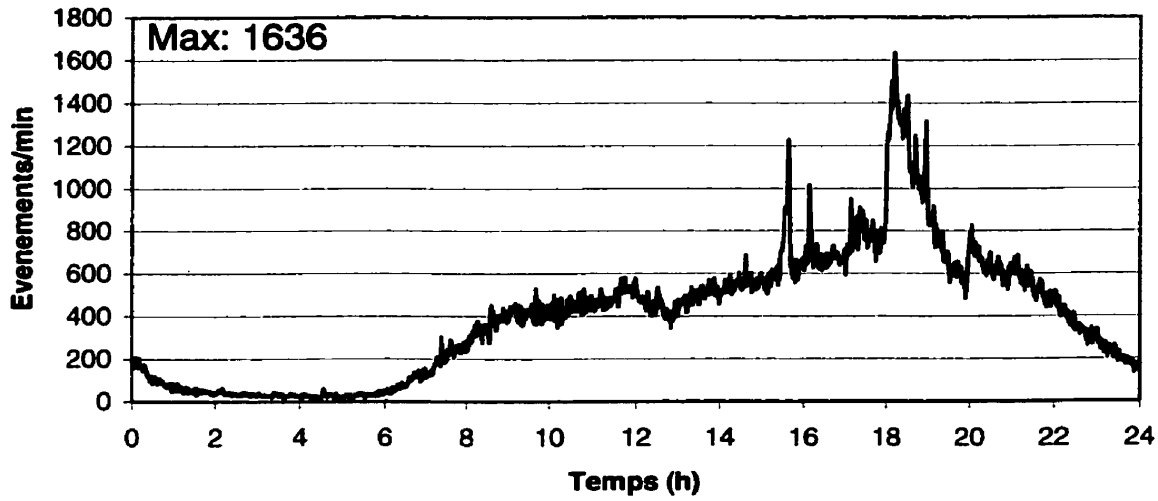

Figure 3.4 Distribution du taux d'arrivée des messages OA

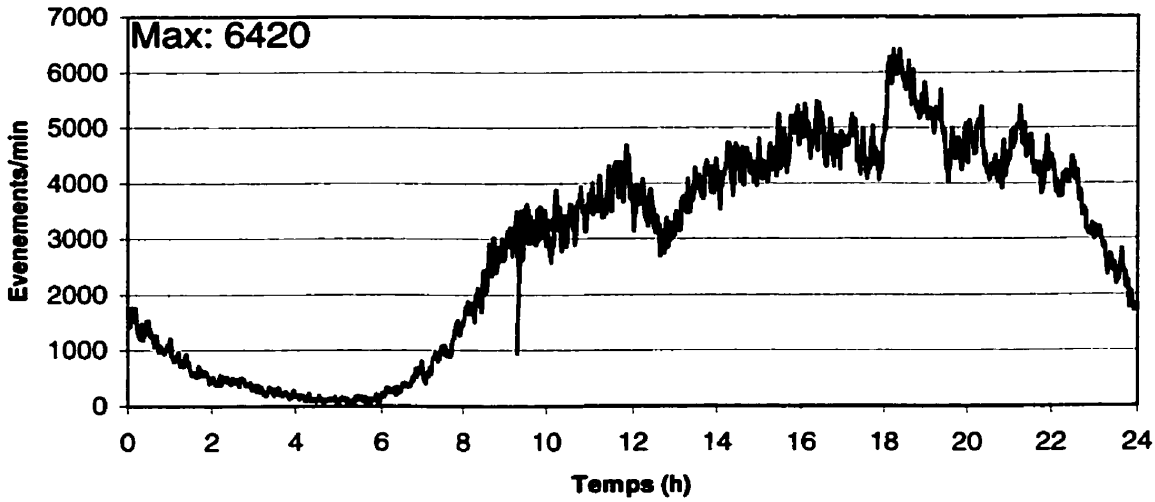

Figure 3.5 Distribution du taux d'arrivée des messages RQM

Les résultats à l'appui, nous pouvons conclure que l'expérience confirme notre design du point de vue des contraintes de base 1 («Traitement externe et minimisation de

l'utilisation de l'UCT») et 5 («Infrastructure peu coûteuse»). Quant à la faisabilité de la contrainte 4 («Traitement en temps «presque» réel»), nous devons nous exprimer avec réserve. Nous avons remarqué que, d'une part, la charge de l'UCT est basse (résultat 4) lors du traitement des événements RQM qui sont relativement agressifs. D'autre part, le temps de latence des événements en général semble être relativement faible (résultat 5). Nous pourrions donc conclure que la contrainte 4 («Traitement en temps «presque» réel») pourra être respectée. Cependant, il nous faut remarquer que la norme ANSI-136 [16] permet le fonctionnement en mode analogique ou en mode numérique et que les terminaux mobiles ne génèrent les messages RQM qu'en mode numérique. La figure 3.6 nous montre les distributions résultantes de l'analyse du type de service requis (contenu dans le message OA). On y voit que le mode analogique domine de loin le mode numérique, ce qui indique que le nombre de messages RQM ne montre pas fidèlement l'activité de l'interface radio. Pour cette raison nos résultats ne nous permettent pas de dire avec assurance qu'il sera possible de respecter la contrainte 4 («Traitement en temps «presque» réel») au niveau de la plate-forme extérieure. Finalement, puisque lors de l'acquisition de tous les événements possibles n'y a eu aucune perte de données et que les événements OA arrivaient dans des délais raisonnables, nous pouvons affirmer avec certitude, qu'au niveau du MSC, notre approche permettra de satisfaire la contrainte 4 («Traitement en temps «presque» réel»). Puisque le temps d'exécution de l'application qui stocke les données peut être considéré comme négligeable. Nous pouvons aussi affirmer avec certitude que nous serons capables d'acquérir les données et les stocker au niveau de la plate-forme d'acquisition.

En somme, nous pouvons conclure que notre architecture respecte les contraintes les plus critiques (1, «Traitement externe et minimisation de l'utilisation de l'UCT» et 5, «Infrastructure peu coûteuse») de manière générale et la contrainte 4 («Traitement en temps «presque» réel») au niveau du MSC. Ceci nous permet de poursuivre notre travail afin de vérifier la possibilité de satisfaire les autres contraintes avec notre approche.

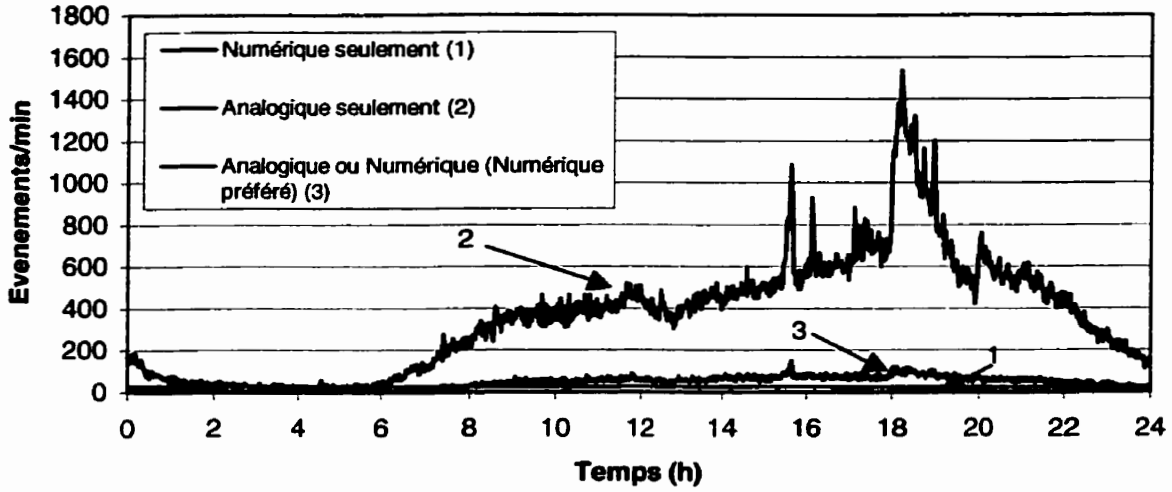

Figure 3.6 Distribution des types de services requis

3.3 Traitement des données

Cette partie constitue le cœur du projet de recherche, la figure 3.7 présente le diagramme de flux de données schématisant ce processus. Nous pouvons y distinguer les trois parties décrites dans les sections précédentes (décodage, tri, reconstruction des séquences). Ces parties sont séparées par deux interfaces (A et B) qui sont constituées par des classes abstraites [31, 32]. Par leur nature ces classes forcent toutes les classes qui en héritent à implanter un certain nombre de méthodes requises. Ceci maximise la flexibilité du système car, par exemple, il devient possible d'utiliser le décodeur avec n'importe quel type d'application qui ne fait qu'hériter de l'interface A.

3.3.1 Décodage des événements

La première partie du flux de données de la figure 3.7 consiste à extraire les événements des fichiers de stockage des groupes d'événements. Le décodeur utilisé

possède une partie générique et une partie générée à partir des fichiers contenants des spécifications en «Data Description Language» (DDL)[1]. Le DDL a été conçu dans le cadre de ce projet. Il permet de décrire le format des messages et d'identifier les champs clés de l'instance d'acquisition. A ce stade nous n'allons pas nous attarder plus sur le décodage des événements car cette partie va être traitée en détails dans les chapitres ultérieurs.

---

[1] Le DDL développé dans ce projet n'a rien en commun avec, le «Data Definition Language» du domaine des bases de données [52], mis à part son acronyme.

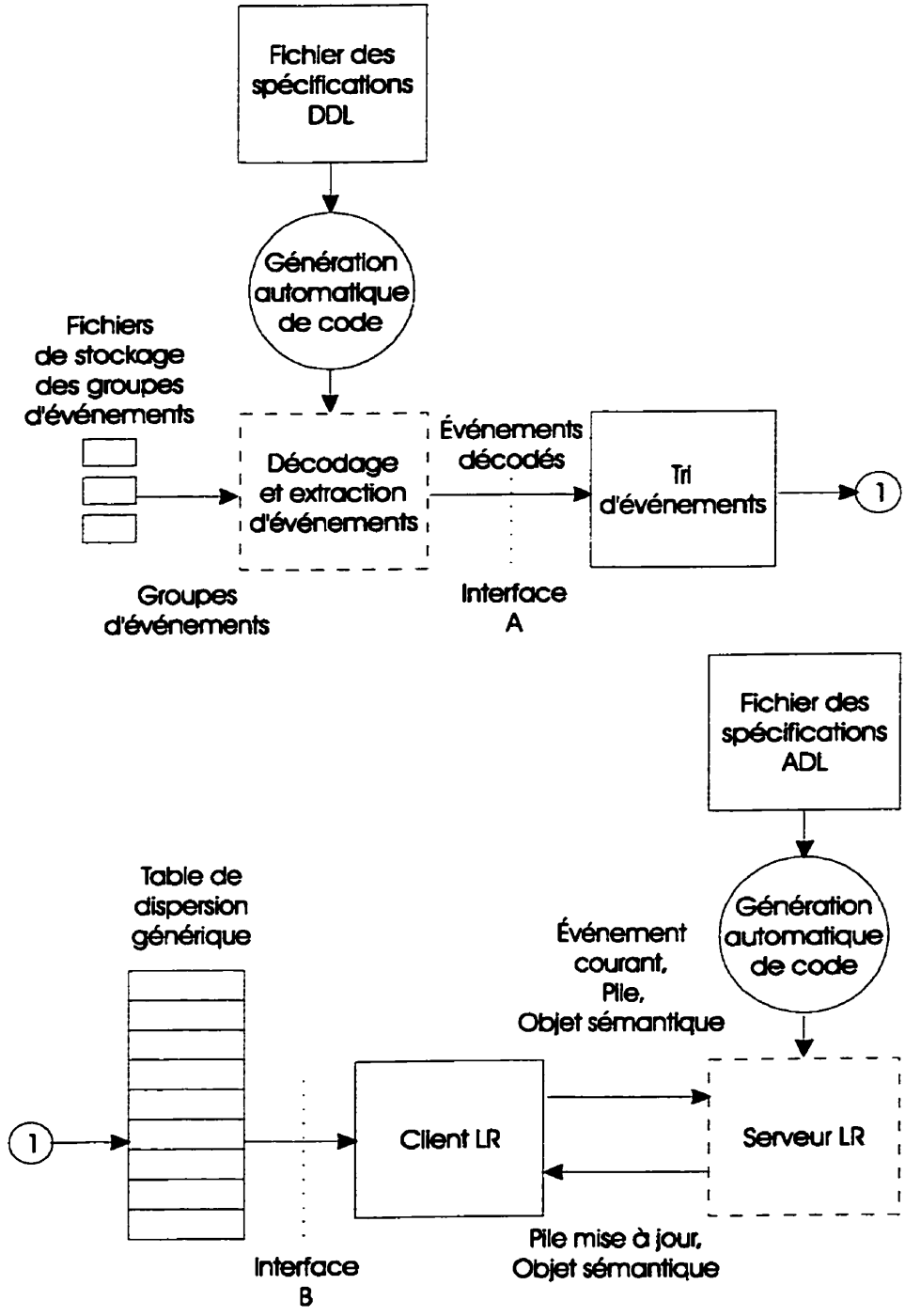

Figure 3.7 Flux de données dans la plate-forme d'acquisition

3.3.2 Tri des événements

La deuxième partie du flux de données de la figure 3.7 consiste à trier les événements selon une clé identifiant l'instance analysée. Les événements reçus par cette partie ont la forme d'objets possédant des méthodes de comparaison et pouvant générer les valeurs de dispersion. Ceci permet de les placer dans une table de dispersion [29, 30]. L'utilisation de la table de dispersion se justifie par une performance d'un ordre constant (O(k)) du point de vue d'exécution et d'un ordre proportionnel à la quantité d'instances analysées (O(n)) du point de vue de l'espace mémoire [29, 30]. Ceci est important pour satisfaire la contrainte 4 («Traitement en temps «presque» réel »). Le contenu de la table de dispersion est constitué des objets «Travailleurs», qui correspondent à la l'interface B. Quand un nouvel événement arrive et ne possède pas de travailleur, un nouveau travailleur est créée en utilisant ce qu'on appelle une «usine de travailleurs» [32]. Le patron de design «usine» («factory») [32] permet de créer de manière spécifique les éléments qui possèdent une interface générique. La figure 3.8 présente en «Unified Modelling Language» (UML) [45] ce patron de design appliqué à notre problème. On peut y remarquer que la table de dispersion utilise une classe abstraite «UsineDesTravaileurs» afin de créer des «Travailleurs» qui vont recevoir les données. En réalité, lors de l'initialisation, «UsineDesTravailleurs» est assignée à l'implantation réelle, qui est «UsineClientLR» et qui génère des «ClientLR»[1], mais qui sont utilisés par la table de dispersion de manière abstraite comme des «Travailleurs».

---

[1] Dans le chapitre 4 nous allons expliquer l'appellation «ClientLR»

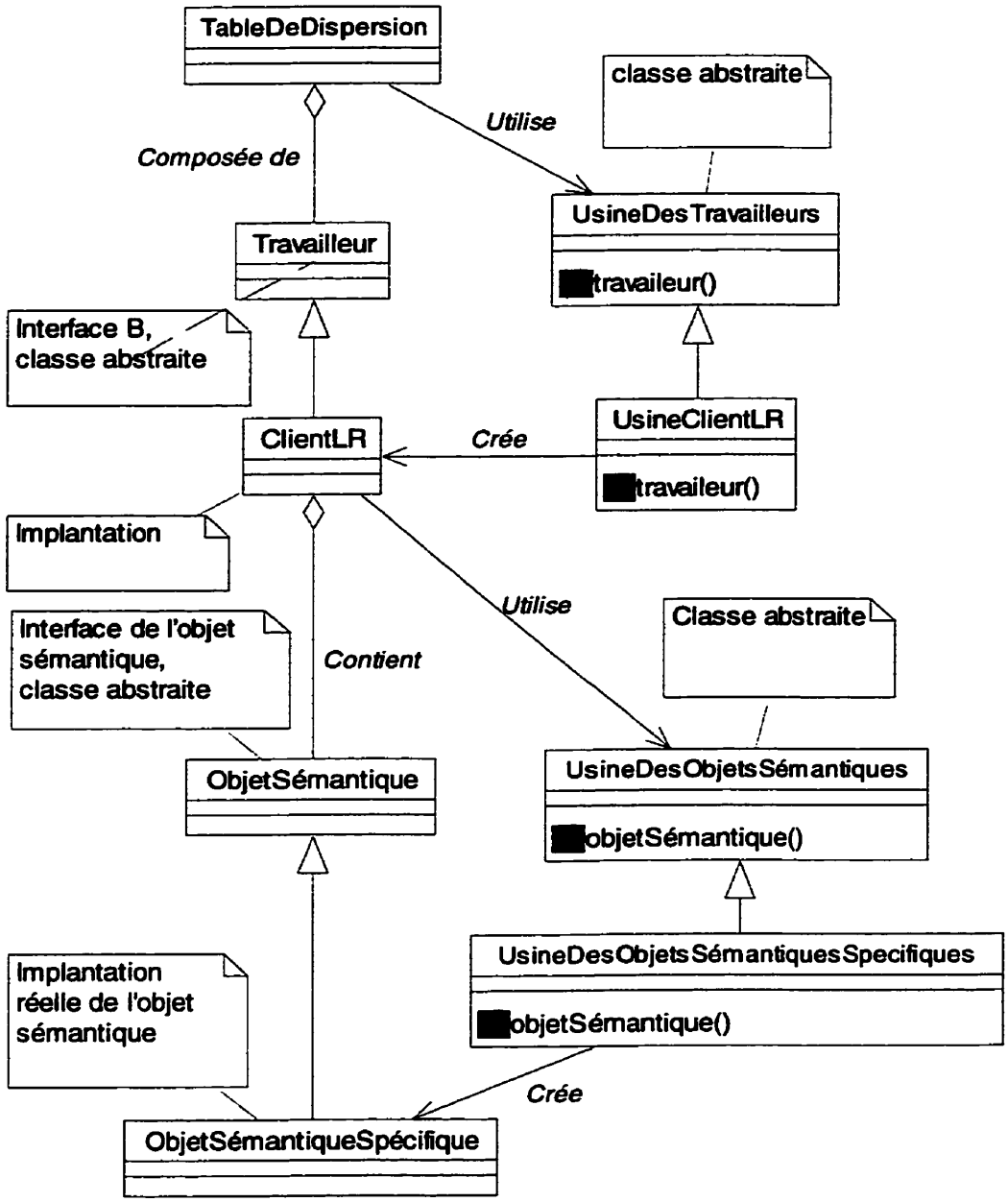

Figure 3.8 Hiérarchie des classes de l'interface B et du client LR

3.3.3 Reconstruction des séquences d'événements

Cette partie constitue la dernière phase du traitement présenté à la figure 3.7. Elle utilise un algorithme d'analyse grammaticale afin de reconstruire les séquences d'événements. Elle est composée de deux éléments principaux : premièrement, les objets «ClientLR» (Clients LR de la figure 3.7), qui contiennent chacun une pile d'analyse grammaticale et un objet sémantique qui constitue l'interface des routines implantées pour les différents types d'analyse. Deuxièmement, le serveur LR, qui contient la machine à états utilisée lors de l'analyse grammaticale. C'est le serveur LR qui va déclencher les actions sémantiques sur les objets sémantiques. Le serveur LR, tout comme le décodeur, contient une partie générique et une partie générée à partir des fichiers de spécifications écrits en «Acquisition Description Language» (ADL). ADL a été conçu dans le cadre de ce projet, il permet de décrire les séquences d'événements, en utilisant une grammaire similaire à la forme de Backus-Naur [24, 27, 34, 35]. Encore une fois à ce stade nous allons nous attarder sur cette partie car elle sera traitée en détails dans les chapitres ultérieurs.

3.3.4 Interface de programmation

Revenons à la figure 3.8. Nous pouvons remarquer que toute la partie implantée par l'utilisateur pour une analyse en donnée est contenue dans les objets sémantiques, qui sont contenus eux-mêmes dans les objets «Travailleur» spécialisés en objets «ClientLR». Cette partie utilise aussi le patron de design «usine» tel que décrit dans 3.3.2. Les objets sémantiques sont créés en même temps que les objets «ClientLR» par une classe que nous appelons ici «UsineDesObjetsSémantiquesSpécifiques» et dont le nom exact est spécifié dans le fichier des spécifications ADL. Les usagers du système doivent donc fournir une classe «UsineDesObjetsSémantiquesSpécifiques» pour que les «ClientsLR» puissent

construire des objets sémantiques permettant d'effectuer les opérations requises par l'acquisition.

3.4 Implantation pratique

Du point de vue pratique, la plate-forme est implantée dans le système d'exploitation Linux en utilisant le langage orienté objet C++. Les parties de génération automatique de code, et plus particulièrement, les compilateurs de DDL et de ADL sont effectués en encapsulant les outils LEX et YACC [33] dans des classes du C++. La modélisation structurale et fonctionnelle en UML est utilisée lors du design orienté objet [45]. Dans ce mémoire nous allons présenter des extraits de cette modélisation en guise d'illustration des approches choisies.

CHAPITRE IV – TRAITEMENT DES DONNÉES

Dans ce chapitre nous allons décrire en détail le cœur du projet. Il s'agit là du traitement des données, de leur décodage jusqu'à leur transfert vers l'application qui effectue les analyses requises. En effet, dans notre présentation nous allons suivre le flux de données présenté dans les chapitres antérieurs. Nous allons commencer par présenter le décodeur ainsi que le langage «Data Description Language» DDL permettant d'effectuer la génération automatique du code. Ensuite nous allons montrer comment le classement par instance d'acquisition est effectué. Finalement, nous allons présenter la partie qui permet de reconstruire les séquences d'événements et qui lance les routines sémantiques, ainsi que le langage «Acquisition Description Language» ADL permettant également d'effectuer la génération automatique du code. Puisque la partie qui effectue la reconstruction des séquences d'événements est beaucoup plus compliquée, nous allons y mettre l'accent.

4.1 Décodeur

Le décodeur d'événements a deux rôles principaux. Premièrement, il doit permettre de transformer les données reçues dans un format propre au MSC en un format utilisable par l'application. Par cet aspect, le décodeur est similaire à l'approche «External Data Representation» (XDR) du système «Remote Procedure Calls» (RPC) [46]. XDR permet de décrire et transformer les données afin de pouvoir les transmettre entre les clients et serveurs implantés sur des plates-formes différentes. La différence majeure entre notre système et le XDR est le fait que dans le système XDR, les clients aussi bien que les serveurs transforment les données en un format normalisé. Il nous est impossible d'utiliser

le même processus à cause de la contrainte 1 («Traitement externe et minimisation de l'utilisation de l'UCT du MSC») qui interdit l'ajout de fonctionnalités complexes au sein du MSC. Pour cette raison, dans notre cas, seul le client effectuera les conversions.

Deuxièmement, le décodeur doit transformer les données en structures assez élaborées pour permettre un tri (par instance d'acquisition) et une reconstruction des séquences de manières génériques. C'est-à-dire, il ne faut pas que des modifications au niveau de l'implantation du décodeur soient requises quand les structures des événements changent. Ceci implique que chaque structure construite doit contenir une identification du type d'événement et une clé permettant de l'associer à une instance d'acquisition.

4.1.1 Format des événements

La plate-forme d'acquisition reçoit les événements par groupes, ces groupes correspondent aux espaces tampons du bloc d'acquisition du MSC. Leur format reflète l'implantation de la norme EIA/TIA-136 au sein du MSC. Cette structure peut changer d'une version du logiciel de contrôle à l'autre. Pour chaque version du logiciel de contrôle un document expliquant la structure exacte est produit. Le document «MDATA events for Package D» de Cobett [15] en est un exemple qui décrit les formats des événements pour la version 4 du logiciel de contrôle. La figure 4.1 présente un exemple du format des événements. Il s'agit d'un bloc de données, similaire aux structures de type champs de bits («bit-fields») en C/C++ [31] ou aux enregistrements compacts («packed records») en Pascal [48]. Nous pouvons distinguer quatre caractéristiques principales :

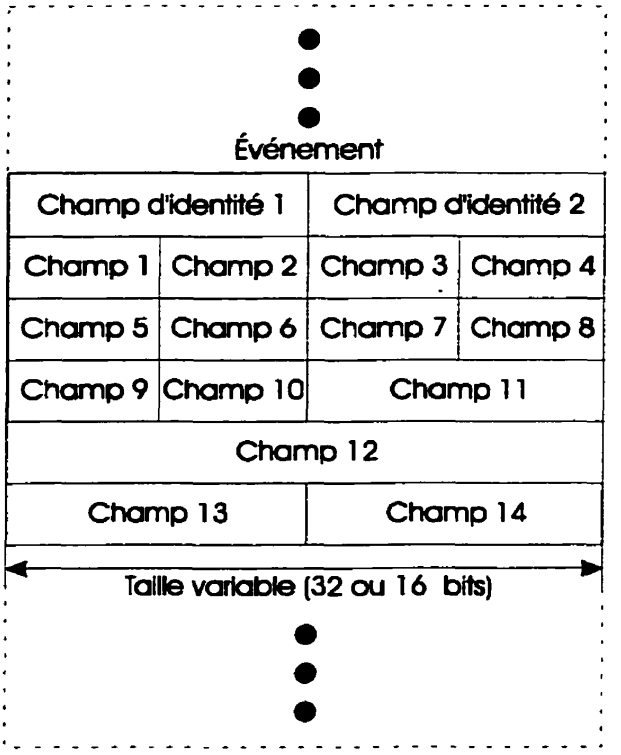

Figure 4.1 Exemple du format d'un événement

1. Chaque type d'événement a une taille fixe.

2. Tous les événements possèdent un en-tête de format fixe qui contient un certain nombre de champs d'identité. Cet en-tête correspond à l'identification interne (MSC) des messages envoyés d'un bloc à l'autre. Nous allons utiliser ces en-têtes afin d'identifier le type de message. Il est intéressant de noter, que dans certaines versions, l'en-tête contient un champ qui indique la taille de l'événement.

3. Les messages peuvent être alignés sur de différentes longueurs du mot, généralement 32 ou 16 bits dépendant du type d'UCT.

4. Toutes les UCT utilisés alignent les mots de données sur les octets les plus significatifs («Big Endian»).

4.1.2 Méthode de décodage

Le décodage des événements se fait en deux ou trois étapes dépendant de la plate-forme physique utilisée pour effectuer l'acquisition.

1. La rotation des octets («Endian swap») est effectuée si requis.

2. Nous obtenons l'identité de l'événement en décodant son en-tête avec un décodeur d'en-têtes commun.

3. À partir de l'identité obtenue, un décodeur d'événements est choisi et le décodage de l'événement se fait champ par champ, chaque type de champ possédant son propre décodeur.

Cette approche nous garantit une flexibilité maximale dans l'avenir. Si jamais des portions d'événements doivent être représentées par de nouvelles structures plus complexes, il sera très facile d'ajouter de nouveaux décodeurs. D'autre part, quand la taille des événements est indiquée dans leurs en-têtes, nous pouvons décoder uniquement le sous-ensemble des événements qui est utile pour l'analyse, le décodage des autres événements pouvant alors être sauté.

4.1.3 Objet événement

Le décodeur d'événements produit ce que nous appelons des objets événements. Un exemple d'un tel objet est présenté à la figure 4.2. Cette figure montre l'objet événement produit à partir de l'événement présenté à la figure 4.1. Nous pouvons y remarquer que l'objet événement est constitué d'un vecteur contenant les différents champs. Chaque élément du vecteur possède une valeur simple contenue dans un entier de 32 bits et un pointeur vers une structure avancée. Un pointeur «NULL» indique que l'entier de 32 bits doit être interprété sinon le pointeur pointe vers une structure avancée. Dans ce cas c'est la structure avancée qui doit être interprétée. Tous les objets événements d'un même type pointent sur une même structure d'identification qui est associée à la structure de la clé d'instance d'acquisition. La structure de la clé est un tableau contenant les index des champs clés dans l'objet événement. De toute évidence, tous les événements doivent définir des clés compatibles, C'est-à-dire que les champs se trouvant sous les mêmes numéros d'index doivent être du même type. Par exemple, ils peuvent tous définir comme clé un tableau d'entiers représentant le numéro du terminal mobile.

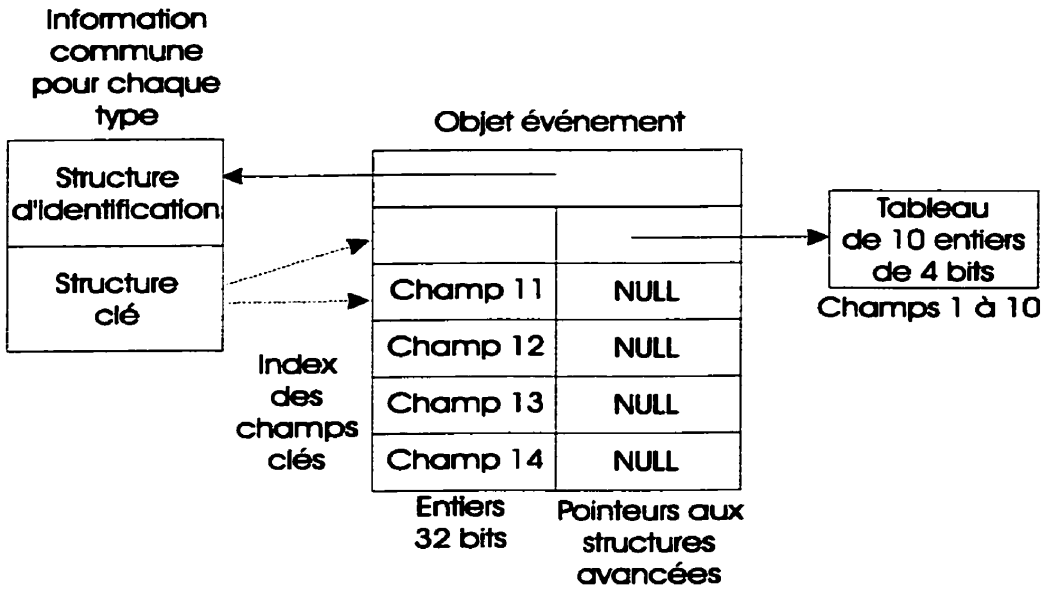

Figure 4.2 Exemple d'objet événement

La figure 4.3 représente la hiérarchie des classes de structures avancées et leur relation avec l'objet événement. Encore une fois, toutes les classes héritent d'une classe abstraite «StructureAvancée» qui sert d'interface et qui impose l'implantation d'un certain nombre de méthodes requises. En particulier, il s'agit des méthodes permettant de cloner (copier) ces structures, de les comparer et de générer les valeurs de dispersions [29, 30]. Toutes ces méthodes seront utilisées plus tard dans le processus de classement par instance d'acquisition. Il faut remarquer que les structures complexes ainsi que les entiers sont encapsulés dans l'objet événement, c'est-à-dire que l'application ne manipule jamais les structures directement, elle fait toujours appel aux méthodes de l'objet événement. Par exemple, quand l'application compare la clé de deux objets événements, elle invoque l'opérateur «==» de l'objet événement. Celui-ci va invoquer les opérateurs «==» des champs faisant partie de la clé des deux objets événements comparés.

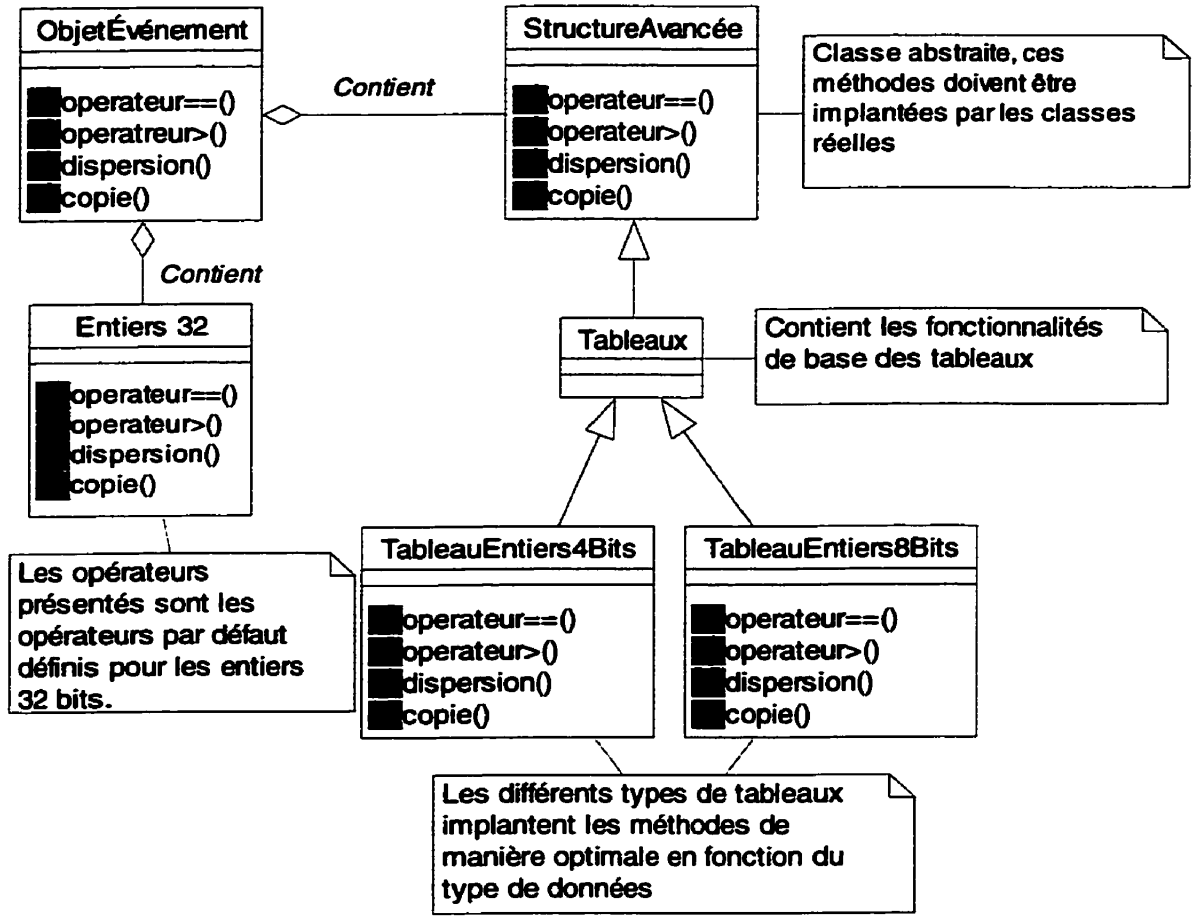

Figure 4.3 Hiérarchie des classes de l'objet événement

4.1.4 Points à améliorer

L'approche prise dans le décodeur pour gérer les objets événements a deux inconvénients principaux.

Premièrement, la partie commune décrivant les caractéristiques de tous les objets événements d'un même type est utilisée uniquement pour garder la structure

d'identification et la structure de la clé. Il serait avantageux d'y garder aussi les informations indiquant les types des champs de l'objet événement. Ceci permettrait de garder une entrée (à la place de deux) par champ de l'objet événement, ce qui réduirait considérablement l'espace requis.

Deuxièmement, du point de vue de l'implantation, les routines d'allocation dynamique du noyau du système d'exploitation sont utilisées, pour créer les objets événements et toutes leurs composantes. Ces routines ont été conçues pour un usage général et peuvent ne pas être optimales du point de vue du temps et de l'espace pour notre application [50]. Comme solution, nous proposons d'allouer à l'avance des ensembles d'objets événements qui seraient saisis et relâchés lors de l'exécution [3]. Ces ensembles pourraient être augmentés ou diminués dynamiquement selon la nécessité [3].

4.1.5 Le langage DDL

Le «Data Description Language» ou DDL est utilisé afin de générer automatiquement la partie du décodeur qui dépend du format des événements. Le DDL contient trois ensembles d'informations. Le premier présente les aspects généraux qui différencient le MSC de la plate-forme d'acquisition tels que la taille ou l'alignement des mots de données. Le deuxième ensemble décrit la structure de l'en-tête des événements en terme de données de base. Finalement, le troisième ensemble décrit le format de tous les événements.

```
architecture {
  endianswap false;
  character 1;
  line 16;
  sizeadj 2;
}

header {
  EID unsigned min 1 max 255 16;
  size NDW unsigned min 1 max 255 16;
}

body {
   event1 2 18 {
      key MIN array (4, 12, 10, 0);
      key CELLID unsigned 16;
      field1 unsigned 32;
      field2 unsigned 16;
      field3 unsigned 16;
   }
}
```

Figure 4.4 Exemple de définitions en DDL

La figure 4.4 présente un exemple pratique de définition en DDL de l'objet événement présenté à la figure 4.2. On peut y distinguer les trois ensembles d'informations présentés sous forme de trois sections.

1) Section «architecture» contient les informations suivantes :
   a) endianswap : indique si la rotation des octets doit être effectuée ou non (appliquer la rotation : « true », ne pas appliquer : « false »).
   b) character : indique la plus petite taille de champ possible (valeurs 1 à la taille du mot).
   c) line : taille du mot (16 ou 32) utilisé pour la rotation des octets.
   d) sizeadj : valeur d'ajustement par rapport à la valeur contenue dans le champ de taille contenue dans l'en-tête.
2) Section «header» (en-tête) contient les définitions des champs qui font partie de tous les événements. Seules les structures simples sont permises (entiers de taille variable).

3) Section «body» (corps) contient la définition de tous les événements.

La définition de chaque événement se fait comme suit :

1) Nom de l'événement.
2) L'identité de l'événement est constituée par une série d'entiers correspondant aux valeurs que peuvent prendre les champs de l'en-tête.
3) Une liste de champs faisant partie de l'événement.

La définition de chaque champ se fait comme suit :

1) Fanion «size», «key» ou aucun fanion.
   a) Un seul champ faisant partie de l'en-tête peut posséder le fanion «size» (taille). Sa valeur va être interprétée comme la taille de l'événement. Dans l'exemple de la figure 4.4 ce champ porte le nom de NDW.
   b) Plusieurs champs faisant partie du corps de l'événement peuvent posséder le fanion «key» (clé). Ceci va indiquer que le champ spécifié fait partie de la clé.
2) Nom du champ.
3) Type
   a) «signed» (signé) : Définit un entier signé, il peut être suivi de limites maximales et minimales. Si une de ces limites est dépassée, une exception est lancée par le décodeur. La définition se termine par la définition de la taille (en bits) de l'entier.
   b) «unsigned» (non signé) : Même chose que signé, mais pour un entier non signé.
   c) «array» (tableau) : Définit un tableau d'entiers non signés avec les paramètres suivants : taille en bits de chaque élément, nombre total d'éléments, valeur maximale permise pour chaque élément, finalement le dernier champ indique si les éléments sont signés un non (0 : non, toute autre valeur oui).

Le tableau 4.1 présente la grammaire du langage DDL décrite à l'aide de la notation Backus-Naur (Backus-Naur Form ou BNF) [25,26,27,34,35]. Il faut remarquer

que lors de ce projet nous utilisons les outils Lex et Yacc [33] afin de générer l'analyseur syntaxique. C'est pourquoi la grammaire doit se plier à certaines contraintes. Par exemple, il nous est impossible d'utiliser la forme étendue de BNF (EBNF) [35] afin de décrire la grammaire de manière plus compacte. Dans notre description nous avons adopté la convention suivante : les productions commencent par une majuscule, les mots réservés et les symboles sont présentés en lettres minuscules et en gras, et les types de base en lettres majuscules. Les types de base incluent les chaînes de caractères (STRING), et les nombres signés (SIGNED) ou non signés (UNSIGNED). Chaque production est suivie d'une description sémantique.

Tableau 4.1 Grammaire du langage DDL

No. de production	Production en format BNF	Signification sémantique
1	Specs : := Architecture_specs Header_specs Body_specs	Définition globale du langage avec ses trois parties : architecture, en-tête et corps de messages
2	Architecture_specs ::= **architecture {** Endian_statement Character_statement Line_statement **}**	Le champ d'ajustement de la taille est optionnel dans la section «architecture»
	| architecture { Endian_statement Character_statement Line_statement Size_adj **}**	

Tableau 4.1 (suite)

No. de production	Production en format BNF	Signification sémantique
3	Hader_specs ::= **header** { Field_list }	La section «header» contient uniquement une liste de champs. Notons que les champs ne peuvent être que des champs simples («signed» ou «unsigned»). Cette contrainte est vérifiée lors de la construction de l'arbre de syntaxe abstrait (Abstract Syntax Tree ou AST) après la phase d'analyse syntaxique.
4	Body_specs ::= **body** { Event_list }	La section «body» contient le corps de messages contenus dans une liste.
5	Event_list ::= Event_list Event_specs | Event_specs	La liste des événements.
6	Event_specs ::= STRING SIGNED { Field_list } | STRING SIGNED SIGNED { Field_list } | STRING SIGNED SIGNED SIGNED { Field_list }	Les événements contiennent la définition des champs. Ils peuvent être identifiés par un, deux ou trois champs d'identité.
7	Field_list ::= Field_list Extended_field_specs | Extended_field_specs	Liste des champs.

Tableau 4.1 (suite)

No. de production	Production en format BNF	Signification sémantique
8	Extended_field_specs ::= **key** Field_specs	Chaque champ peut être défini comme un champ «key» (clé) ou un champ «size» (indiquant la taille). Notons que les champs «key» peuvent uniquement faire partie du corps du message et un seul champ «size» peut faire partie de l'en-tête. Ces deux conditions sont vérifiées lors de la construction de l'AST.
	| **size** Field_specs	
	| Field_specs	
9	Field_specs ::= STRING **signed** Min_max_specs UNSIGNED;	Les définitions des champs incluent les trois types de données actuellement implantés «signed» (entier signé), «unsigned» (entier non-signé) et «array» (tableau). Notons que le type tableau, en effet, représente plusieurs sortes de tableaux dépendant de la configuration, par exemple des tableaux de 4, 8 ou 16 octets.
	| STRING **signed** UNSIGNED;	
	| STRING **unsigned** Min_max_specs UNSIGNED;	
	| STRING **unsigned** UNSIGNED;	
	STRING **array** (UNSIGNED, UNSIGNED, SIGNED, SIGNED);	
10	Min_max_specs ::= **max** SIGNED	Les définitions des valeurs limites contenues dans les variables.
	| **min** Signed	
	| **max** SIGNED **min** SIGNED	
11	Endian_statement ::= **endianswap true;**	Définition de la spécification de la rotation des octets.
	| **endianswap false;**	
12	Character_statement ::= **character** UNSIGNED;	Définition de la taille minimale d'un champ.
13	Line_stetement ::= **line** UNSIGNED;	Définition de la taille des mots.
14	Size_adj ::= **sizeadj** SIGNED;	Définition de l'ajustement de la taille.

4.1.6 Décodage et génération automatique de code

La figure 4.5 présente la structure des classes faisant partie du décodeur. Cette structure contient deux classes abstraites [31,32] «Décodeur» et «DécodeurÉvénement». La classe «Décodeur» coordonne le processus du décodage, elle contient le décodeur d'en-têtes, et la table de dispersion qui contient à son tour les décodeurs d'événements. Cette classe définit une méthode de configuration abstraite. La classe «DécodeurÉvénement» contient les objets permettant le décodage des champs de base, ainsi que les décodeurs permettant le décodage des objets avancés. Elle est responsable du décodage de tous les événements. Cette classe définit aussi une méthode de configuration abstraite. Deux types de classes sont générés à partir du DDL. Le premier type, est la classe «DécodeurGénéré» qui hérite de la classe «Décodeur». Cette classe est générée une seule fois. Elle ne fait qu'implanter la méthode de configuration qui initialise le décodeur d'en-têtes (classe «DécodeurEnTête») et remplit la table de dispersion (classe «TableDeDispersion») avec les instances de la classe «DécodeurÉvénementGénéré». Le deuxième type de classe est la classe «DécodeurÉvénementGénéré» qui hérite de la classe «DécodeurÉvénement». Elle est générée plusieurs fois, une fois pour chaque type d'événement. Cette classe implante aussi la méthode de configuration qui permet de configurer la séquence de décodage pour chaque événement.

L'application qui utilise le décodeur ainsi généré crée une instance de la classe «DécodeurGénéré» et lance la méthode de configuration. Cette méthode va créer et initialiser toutes les instances de la classe «DécodeurÉvénementGénéré». Après la phase de configuration la structure est prête pour procéder au décodage. Le décodage consiste à fournir le contenu d'un espace tampon contenant un groupe d'événements reçus au «DécodeurGénéré». Ensuite, l'application doit appeler plusieurs fois la méthode permettant d'extraire les événements, jusqu'à ce que celle-ci retourne un pointeur «null». Cela indique l'épuisement de données de l'espace tampon.

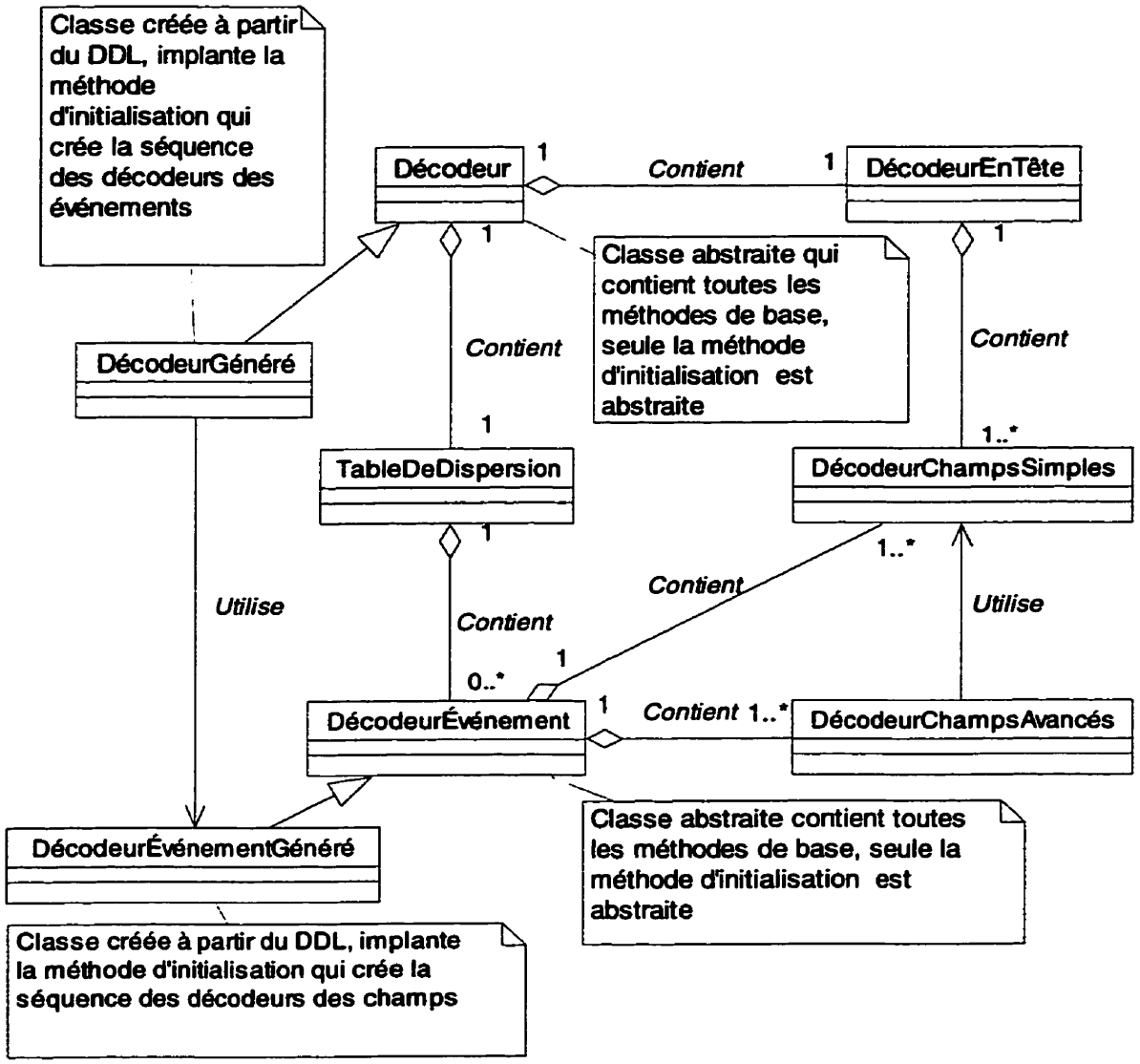

Figure 4.5 Hiérarchie de classes du décodeur

4.2 Classement par instance d'acquisition

Une fois que les objets événements sont créés, tel que présenté dans le chapitre précédent, ils doivent être passés à l'objet «Travailleur» (présenté dans le chapitre 3) responsable de cette instance d'acquisition. Cette partie est exécutée par la table de dispersion qui contient les objets «Travailleur». Dans un premier temps la table de dispersion invoque la méthode «dispersion» sur l'objet événement, celui-ci invoque à son tour les méthodes «dispersion» sur tous les champs faisant partie de la clé. Les résultats sont fusionnés en une seule valeur, au sein de l'objet événement à l'aide d'un opérateur «ou exclusif». Selon Knuth [29] et Sedgewick [30] l'opérateur ou exclusif s'applique bien pour la génération de valeurs de dispersion. La valeur de dispersion sert à déterminer l'entrée dans la table de dispersion. Si jamais cette entrée est déjà utilisée par un autre «Travailleur», selon les méthodes présentées par Knuth [29] et Sedgewick [30], une table est créé pour contenir les collisions. Si la table de collision contient plusieurs éléments alors l'opérateur «==» de l'objet événement est utilisé afin de trouver le bon «Travailleur» ou d'en créer un nouveau s'il s'agit d'un premier événement pour cette instance d'acquisition. Dans le cas ou un nouveau «Travailleur» est créé l'objet événement produit un autre objet événement qui contient uniquement la copie des champs clés et l'objet. Cet objet est utilisé par le «Travailleur» afin d'identifier l'instance d'acquisition desservie par ce «Travailleur». Comme nous pouvons le remarquer, ce processus est entièrement générique, grâce à l'encapsulation [45] des champs au sein des objets événements. De cette manière, aucune modification du code n'est requise quelle que soit la structure des événements et quelle que soit la clé d'acquisition.

4.3 Reconstruction des séquences

La reconstruction des séquences d'événements est effectuée en partie par les objets «ClientLR» introduits dans le chapitre 3. Dans cette partie, nous allons présenter notre approche dans la résolution de ce problème.

4.3.1 Problématique de la reconstruction des séquences d'événements

La reconstruction des séquences d'événements est la partie la plus importante de ce projet. En effet, la contrainte 2 du chapitre 2 «Corrélation et reconstruction des séquences» y est exclusivement dédiée. Cette partie implante l'interface de programmation qui doit être très souple afin de respecter la contrainte 3, («Facilité, flexibilité et ouverture pour d'autres»). Évidemment, comme tout le reste du projet cette partie doit aussi être implantée de manière efficace car il nous faut offrir un traitement en temps presque réel (contrainte 4, «Traitement en temps presque réel»).

4.3.1.1 Norme ANSI-136

La norme ANSI-136 [16] définit entre autres les 3 couches du protocole permettant les communications mobiles. Les couches 1 et 2 sont utilisées uniquement afin d'acheminer les messages entre les terminaux mobiles et les stations d'antennes [16, 36]. C'est pourquoi elles sont complètement transparentes pour la plate-forme d'acquisition. La norme ANSI-136 définit la machine à états globale d'un terminal. Cette machine à états est composée de plusieurs machines à états de la couche 3. Chacune effectue une procédure dans l'établissement et le maintien d'un appel [36]. Par exemple, un terminal mobile effectuant un appel doit d'abord saisir un canal de voix, ce qui constitue une procédure qui demande un échange de plusieurs messages. Ensuite, ce terminal mobile va rester connecté

tout en se déplaçant. Chaque fois qu'il effectuera un changement de cellule il va démarrer une autre procédure qui requerra d'autres échanges de messages entre le terminal mobile et le MSC. Cette procédure pourra se répéter jusqu'à la terminaison de l'appel ou jusqu'au départ du mobile du secteur desservi par le MSC en faveur d'un autre MSC. Nous pouvons conclure que la machine à états globale est en quelque sorte hiérarchisée ou modulaire. Il est possible que certains échanges d'événements surviennent dans plus d'un état de la machine à états globale. La figure 4.6 représente un exemple d'une telle machine à états qui contient trois états dont chacun possède deux procédures. Ces procédures constituent des «sous-machines» à états. Tel que nous pouvons le remarquer, la procédure «A» peut être lancée au sein de l'état 1 aussi bien qu'au sein de l'état 3. Dans la norme ANSI-136 la réception d'un appel constitue un exemple de cette caractéristique. Un appel peut être reçu pendant que le terminal mobile campe sur un canal de contrôle (inactif), aussi bien que quand il est en train d'exécuter une procédure d'enregistrement périodique [16, 36].

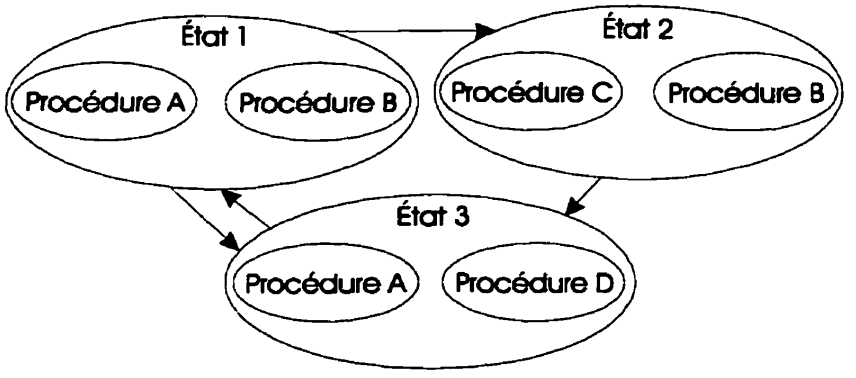

Figure 4.6 Machine à états hiérarchisée

4.3.1.2 Analyses inspirées du «Data Mining»

Il existe une forme d'analyse appelée le «Data Mining» [4] qui consiste à manipuler de grandes quantités de données afin de trouver les relations parmi ces données

[4, 5, 6, 7, 8, 9, 10]. Une technique fréquemment utilisée est la technique de généralisation [4]. L'analyse génère alors les résultats à différents niveaux d'abstraction. Par exemple, si nous prenons une séquence d'événements et nous construisons une table d'appels contenant toutes les informations, nous pouvons dire que cette table d'appels constitue le niveau d'abstraction ou de généralisation le plus bas. Ensuite, si nous faisons des analyses statistiques sur, par exemple, le type d'appel (analogique versus numérique), et les échecs des appels, nous effectuons une généralisation [4]. Ensuite nous pourrions continuer cette généralisation en calculant uniquement la proportion d'échecs par rapport au nombre total d'appels. Des structures de ce genre sont aussi appelées cubes ou cônes de données [4]. Il est intéressant de noter que ce type d'analyse va de pair avec la notion de la machine à états hiérarchisée. C'est pourquoi Askerup [47] suggère d'utiliser les techniques du «Data Mining» pour analyser le comportement du réseau.

4.3.1.3 Aspects dynamiques

L'expérience préliminaire (chapitre 3) nous a montré que l'activité de l'interface radio produit une quantité très importante de données. Il est très difficile de manipuler de telles quantités de données. Par exemple, il est pratiquement impossible de les stocker dans une base de données (les événements de types différents dans des tables différentes) et d'effectuer des corrélations. Chaque corrélation d'une table avec une autre table requiert d'effectuer le produit cartésien [52]. Le temps d'exécution du produit cartésien peut augmenter de manière géométrique avec la quantité de données contenues dans les tables. Ceci nous pousse à dire qu'il serait avantageux d'effectuer les corrélations requises au fur et à mesure que les données sont acquises. Quand le système fonctionne, seule la partie de données relatives aux appels actifs est valide. Les données des appels terminées peuvent alors être ignorées. Il faut aussi ajouter que parmi les appels actifs il est intéressant d'extraire seulement certains scénarios particuliers. Par exemple d'analyser seulement les appels qui requièrent le mode numérique ou qui font appel à des fonctionnalités

particulières. Dans de nombreux cas, pendant de l'acquisition d'une séquence, nous pouvons nous rendre compte qu'un certain nombre de critères n'est pas respecté. Par conséquent, il ne sert à rien de continuer d'acquérir cette séquence. Dans ce projet nous voulons exploiter d'avantage l'aspect dynamique du problème en reconstruisant les séquences d'intérêt au fur et a mesure que les données sont reçues par le système. Nous voulons que cette acquisition se fasse selon certaines règles prédéfinies.

4.3.2 Utilisation des algorithmes d'analyse syntaxique

Dans la section précédente nous avons présenté les différents aspects du problème de la reconstruction des séquences d'événements. Si nous nous attaquons au problème en nous attardant à la génération des événements, nous pouvons remarquer que le MSC est un système de contrôle par événements discrets. Il peut être modélisé comme un générateur de langage formel [22]. Dans ce projet, nous nous intéressons uniquement à la partie du MSC qui régit l'interface radio. Toutes les règles de fonctionnement de l'interface radio sont définies par la norme ANSI-136 [16]. Évidemment, les séquences des événements acquis par notre plate-forme doivent respecter ces règles. C'est pourquoi, nous nous proposons d'utiliser un algorithme d'analyse syntaxique auquel nous fournirons un ensemble de règles de grammaire qui décriront le fonctionnement de l'interface radio. Les événements vont constituer les symboles terminaux. En vertu de ce qui a été dit dans la section précédente, cet ensemble va en effet constituer un sous-ensemble de toutes les spécifications ANSI-136. Ce sous-ensemble va représenter la partie de la norme qui sera analysée.

Le processus d'analyse syntaxique est divisé en deux parties. La première partie est purement une analyse des séquences des symboles terminaux du langage. La deuxième partie est la partie sémantique. C'est là que la signification des ces symboles terminaux et leurs séquences est examinée [24, 25, 26, 27]. Généralement, les générateurs d'analyseurs

grammaticaux [33] génèrent la partie d'analyse syntaxique et laissent aux usagers implanter la partie d'analyse sémantique [24, 26, 27]. En effet, cela permet de définir une interface de programmation propre et bien isolée [24, 33]. Au cours de l'exécution, la partie sémantique (implantée par l'usager) se fait solliciter par la partie générique avec des séquences d'éléments du langage.

Dans la littérature nous avons trouvé deux exemples de systèmes utilisant l'approche d'analyse syntaxique de manière similaire. Le premier système est proposé par J. Yost et al [12]. Il permet de décrire à l'aide d'un langage simple certaines tendances ou caractéristiques de séquences de données. Les séquences de données respectant ces caractéristiques vont être représentées graphiquement. Ce système permet essentiellement d'extraire des séquences d'intérêt à partir d'un très grand ensemble de données. Le deuxième système est proposé par A. F. Bobick et Y. A. Ivanov [19]. Ce système utilise l'analyse syntaxique probabiliste pour, par exemple, reconnaître les gestes de la main. Dans leur exemple les auteurs utilisent des connaissances à priori pour créer une grammaire qui décrit la signification des séquences de mouvements de la main. Le système que nous proposons est très similaire à ces deux applications. Dans notre cas, tout comme dans le premier exemple, il s'agit de détecter des séquences d'événements correspondant à l'activité de l'interface radio afin de pouvoir l'observer et analyser son comportement.

4.3.3 Classe de langage

Nous avons proposé d'effectuer la reconstruction des séquences d'événements en utilisant les algorithmes d'analyse syntaxique. Tout d'abord, il nous faut déterminer le type de grammaire dont il s'agit. De manière plus générale, il faudra déterminer le niveau de la hiérarchie de Chomsky auquel se rapportent les spécifications de la norme ANSI-136.

Nous sommes intéressés par le fonctionnement des protocoles de la couche 3 faisant partie de la machine à états globale. Le fonctionnement de la couche 3 est similaire au fonctionnement du protocole à bit alterné [44], c'est-à-dire qu'une entité envoie des requêtes vers une autre entité, et s'attend à recevoir une réponse positive ou négative. Les requêtes aussi bien que les réponses peuvent être perdues si les liens non fiables sont utilisés. C'est pourquoi toutes les requêtes sont protégées par des minuteries. Ce fonctionnement peut être représenté par une machine à états finis sans mémoire [44]. Les machines à états finis sans mémoire peuvent être décrites par des grammaires de type 3 [26, 27]. Ce type de grammaire est aussi appelé grammaire régulière. Dans notre cas nous avons une composition de plusieurs machines à états qui forment une machine à états globale. Les différentes machines à états représentant les procédures de la couche 3 peuvent être invoquées dans différents états de la machine à états globale. Ceci implique que la machine à états globale requiert de la mémoire afin de pouvoir suivre les niveaux d'imbrication. Ceci nous force à utiliser le type 2 de la hiérarchie de Chomsky, afin de pouvoir décrire la machine à états globale. Le type 2 correspond aux grammaires sans contexte [26, 27]. Ces grammaires peuvent être analysées par des machines à états finis possédant une pile qui sert alors de mémoire et qui permet de retenir le niveau d'imbrication [24,25, 26, 27]. La plupart des analyseurs de grammaires sans contexte possèdent la capacité de prendre les décisions en tenant compte de un ou plusieurs symboles d'avance («look ahead symbols»), c'est-à-dire qu'en cas d'ambiguïté syntaxique l'analyseur va attendre qu'un ou plusieurs symboles supplémentaires arrivent avant de pouvoir reconnaître une production. Ces symboles vont permettre de décider quelle action doit être prise. L'utilité de cette approche est d'enlever les ambiguïtés d'une grammaire et, par conséquent, de permettre d'analyser un plus grand ensemble de grammaires. Nous avons montré que les machines à états décrites par la norme ANSI-136 peuvent être décrites par une grammaire régulière. Les grammaires régulières ne requièrent pas de symbole d'avance [24, 25, 26, 27]. Dans notre cas, la seule raison pour laquelle nous utilisons une grammaire plus élaborée que la grammaire régulière vient du besoin de

mémoriser l'état de la machine à états globale. C'est pourquoi, nous n'aurons pas besoin d'implanter de techniques utilisant les symboles d'avance.

4.3.4 Choix de méthode d'analyse syntaxique

À ce stade nous avons déterminé que nous pouvons décrire les séquences d'événements par une grammaire sans contexte. Maintenant, il nous faudra implanter un générateur d'analyseur syntaxique capable de reconstruire les séquences d'événements décrits à l'aide d'une telle grammaire. Ici nous avons essentiellement le choix entre deux approches possibles.

4.3.4.1 Méthode LR(k)

La première approche possible est l'approche du bas vers le haut («Bottom up»). Cette approche consiste reconstituer les productions à partir des productions les plus simples et de monter de niveau au fur et à mesure que les nouveaux éléments sont reçus. Les analyseurs grammaticaux qui implantent cette technique sont identifiés comme des analyseurs grammaticaux de type LR(k). «L» veut dire que les éléments du langage sont interprétés de gauche vers la droite («left to right»). «R» veut dire que les productions sont dérivées de droite en premier dans l'ordre inversé (on remonte les productions à partir du bas ou à partir des productions les plus simples) («rightmost derivations in reverse»). Finalement, «k» indique le nombre de symboles d'avance. Les analyseurs grammaticaux LR(k) sont relativement difficiles a construire. L'algorithme se base sur une matrice qui dans la plupart des cas est générée automatiquement à partir des spécifications. L'algorithme de la méthode LR(k) s'exécute en temps linéairement proportionnel au nombre de symboles terminaux du langage.

4.3.4.2 Méthode LL(k)

La deuxième approche possible est celle du haut vers le bas («Top down»). Cette approche consiste à reconstituer les productions en partant du symbole de départ d'une grammaire et en descendant les productions. Les analyseurs grammaticaux qui implantent cette technique sont identifiés comme des analyseurs grammaticaux de type LL(k). Le premier «L» veut dire que les éléments du langage sont interprétés de gauche vers la droite («left to right»). Le second «L» veut dire que les productions sont dérivées de gauche en premier («leftmost derivations»). Finalement, «k» indique le nombre de symboles d'avance. Les analyseurs grammaticaux LL(k) sont relativement faciles à construire. Ils sont souvent présentés comme des analyseurs grammaticaux prédictifs [24, 26, 27], c'est-à-dire qu'en lisant les symboles terminaux, l'analyseur essaie de prévoir quelle production sera utilisée à partir de la production dans laquelle il se trouve présentement. L'algorithme de la méthode LL(k), tout comme celui de la méthode LR(k), s'exécute aussi en temps linéairement proportionnel au nombre de symboles terminaux du langage. Les analyseurs LL(k), contrairement aux analyseurs LR(k), peuvent être implantés assez facilement directement sans l'utilisation d'outils de génération automatique. D'autre part, puisque l'analyse se fait de haut en bas, les actions sémantiques sont déclenchées dès que l'analyseur syntaxique réalise qu'un certain symbole non terminal va être généré [26]. Ceci garantit une exécution des actions sémantiques dès que les symboles terminaux arrivent dans le système, ce qui est très intéressant du point de vue de l'exécution en temps «presque» réel.

4.3.4.3 Choix de la méthode LR(k)

Tel que nous l'avons dit l'analyse syntaxique avec méthode LL(k) est très facile à implanter, en fait beaucoup plus facile que l'analyse syntaxique avec la méthode LR(k) qui requiert la génération de matrices complexes [24, 26]. Cependant, la partie sémantique de

l'analyse dans le cas de la méthode LL(k) est beaucoup plus complexe [24, 26]. Cela provient du mode de fonctionnement de cet algorithme. En effet, tel que nous l'avons dit dans la section précédente les actions sémantiques sont déclenchées dès que l'analyseur syntaxique réalise qu'un certain symbole non terminal va être généré [26]. Il faut donc que l'usager spécifie des actions sémantiques non pas, comme avec l'analyseur LR(k), pour chaque production, mais pour chaque symbole non terminal présent dans ces productions. Dans ces actions sémantiques l'usager doit propager (par l'héritage ou synthèse [24, 26]) les valeurs sémantiques des différents symboles terminaux afin de réaliser les actions désirées. Parsons [24] parle même de la nécessité, dans certains cas, d'implanter une pile sémantique indépendante de la pile d'analyse syntaxique. Nous voyons donc qu'avec une approche qui se base sur les algorithmes LL(k) les usagers du système doivent être au courant du fonctionnement de la méthode d'analyse syntaxique. Cela constitue un défaut majeur par rapport à la méthode LR(k). La méthode LR(k) détecte quand une séquence survient et déclenche alors une action sémantique en lui fournissant tous les symboles terminaux et non terminaux faisant partie de la production. Par sa nature, LL(k) rend impossible l'implantation de l'interface usager telle que décrite dans 4.3.8, et qui est un des requis de notre solution (contrainte 3 «Facilité, flexibilité et ouverture pour d'autres applications»).

D'autre part, par sa nature, la méthode LL(k) ne permet pas la récursivité à gauche. Il est donc impossible d'avoir des productions qui, par exemple, représentent des listes, et qui ont la forme «L ::= L E | E». En effet, ces productions doivent être transformées en deux productions «L :: = E X» et «X ::= L | ∈» où «∈» représente une production nulle. Ceci dans notre cas constitue un problème assez grave qui provoque une dégénérescence des arbres d'analyse syntaxique («parse tree»). Un exemple de dégénérescence est présenté à la figure 4.7. Cette dégénérescence se traduit par une croissance excessive de la pile d'analyse syntaxique. Dans les applications habituelles, telles que les compilateurs, ceci ne cause aucun tord car en règle générale les listes ont des tailles modérées.

Malheureusement, dans notre cas, il existe plusieurs situations où les terminaux mobiles peuvent envoyer des quantités importantes d'événements périodiques. Un exemple d'un tel événement est le «Radio Quality Message» (RQM) présenté dans le chapitre 3. Puisque de nombreux usagers risquent de générer ces messages simultanément, la plate-forme d'acquisition risquerait de manquer de mémoire si la méthode LL(k) était employée.

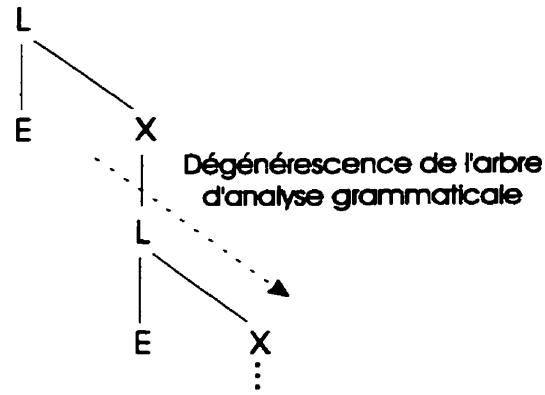

Figure 4.7 Dégénérescence de l'arbre d'analyse syntaxique avec la méthode LL(k)

De son côté, la méthode LR(k) commence à analyser les séquences à partir des productions les plus simples. Ces productions vont représenter les parties de la machine à états les plus élémentaires et donc situées le plus bas dans la machine à états globale. Il sera alors facile de construire des arrangements de données qui vont permettre d'effectuer les généralisations et effectuer des analyses inspirées du «Data Mining».

Cependant, la méthode LR(k) n'est pas parfaite non plus. En effet, son fonctionnement nécessite deux étapes principales : le décalage et la réduction [24, 26]. Cela implique que les symboles qui arrivent à l'entrée du système doivent d'abord être mis sur la pile pour ensuite être réduits par une production. Aucun symbole ne peut être réduit dès qu'il entre dans le système. La réduction peut se faire seulement lors de l'arrivée d'un autre symbole. Cela peut avoir des implications néfastes au niveau de l'exécution en temps «presque» réel des règles sémantiques associées à certaines productions de la grammaire,

par exemple, dans une grammaire qui décrit un appel comme un signal de départ, suivi par une série de procédures de changements de cellules, suivies à leur tour par un signal de terminaison d'appel. L'action sémantique associée à chacun de ces changements ne sera pas exécutée avant que le changement suivant ou la terminaison d'appel ne surviennent. Bien entendu, une solution facile serait de décrire la grammaire en y ajoutant des messages périodiques comme le message «RQM» décrits dans le chapitre 3. Dans ce cas, après le changement de cellule, l'action sémantique serait déclenchée dès l'arrivée du premier message «RQM». Précisons finalement que ce phénomène ne concerne que certaines productions et que la production de départ n'est jamais affligée de ce problème.

Nous voyons qu'il n'existe pas de méthode parfaite pour résoudre notre problème. Cependant, la méthode LL(k) présente trop de désavantages pour notre application par rapport à la méthode LR(k). C'est pourquoi nous avons décidé de choisir cette dernière pour effectuer la reconstruction des séquences d'événements.

4.3.5 Implantation de l'algorithme LR(0).

Nous avons choisi d'utiliser la méthode LR(0), qui correspond à la méthode LR(k) sans symbole d'avance. L'algorithme du LR(0) se base sur une matrice d'exécution. L'algorithme principal est indépendant de la grammaire. Seulement la matrice doit être générée pour chaque grammaire. Il est intéressant de remarquer que l'algorithme et la structure de la matrice sont indépendants du nombre de symboles d'avance («k»). Seuls l'algorithme de génération de la matrice et son contenu changent. La figure 4.8 tirée de la référence [24] présente une implantation typique d'un analyseur syntaxique du type LR(k). Nous pouvons y distinguer quatre éléments principaux :

1. L'entrée de données. Dans notre cas elle est constituée par le décodeur et l'application de tri.
2. La pile qui mémorise les états et les symboles reçus.

3. La matrice d'actions et sauts qui dirige l'exécution de l'algorithme.
4. L'algorithme principal qui reçoit les symboles à partir de l'entrée.

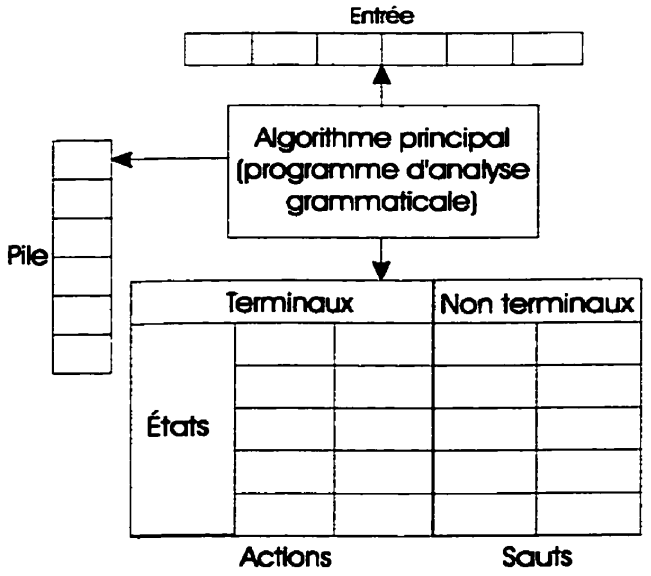

Figure 4.8 Structure d'un analyseur syntaxique

En se basant sur l'état qui se trouve sur la pile et le dernier symbole reçu, l'algorithme exécute les actions spécifiées dans la matrice. Si jamais la séquence ne correspond pas à la grammaire, ou si une action sémantique retourne un code d'erreur, l'analyse sera arrêtée. Normalement, dans ces circonstances l'entrée est arrêtée. Dans notre cas il est impossible d'arrêter l'arrivée des données. Ceci arrêterait l'arrivée des données pour toutes les autres instances d'acquisition. Si l'activité de l'entité d'acquisition n'est pas terminée (e.g. la connexion du terminal mobile n'est pas terminée) d'autres données vont être reçues par le système pour cette instance. Chacune de ces données va lancer le démarrage d'une nouvelle analyse syntaxique pour cette instance d'acquisition. Très probablement, cette analyse échouera aussitôt car le symbole reçu ne correspondra pas à un symbole valide pour le démarrage de l'analyse. Afin de résoudre ce problème, lors du démarrage d'une analyse syntaxique, nous devons vérifier si le symbole reçu correspond à

un symbole faisant partie de l'ensemble des premiers généré à partir du symbole de départ de la grammaire [24].

Enfin, un dernier problème potentiel est le fait que notre plate-forme pourrait ne pas toujours être avertie de la terminaison d'un appel. Par exemple, l'usager peut définir un appel par une séquence commençant avec un signal de début et se terminant avec un signal de terminaison. Cependant, il est aussi possible que la pile électrique du terminal mobile s'épuise et que le terminal mobile perde la communication sans avertir le système. Il est aussi possible (et très probable) que le terminal mobile quitte le secteur contrôlé par le MSC au profit d'un autre MSC. Dans ces deux cas, d'autres types de messages seront générés. Puisqu'ils ne font pas partie de notre acquisition, ils seront éliminés au niveau du décodeur. Dans ces circonstances, le système va rester en attente du dernier message qui n'arrivera probablement jamais. Pire encore, à la place du message de terminaison nous pourrions recevoir un autre message du début d'une autre communication initié par le même terminal mobile. La seule façon que nous avons de nous protéger contre ce phénomène est de démarrer des compteurs à rebours pour chaque reconstruction de séquence. Si jamais une séquence ne change pas d'état après cette période sa reconstruction de séquence sera terminée.

4.3.5.1 Contenu de la matrice

La matrice est essentiellement composée de deux parties, une partie qui contient les actions et une autre partie qui contient les spécifications des sauts. Bien entendu, les deux parties doivent être générées à partir des spécifications syntaxiques. Chaque ligne de la matrice représente un état de l'analyseur, chaque colonne correspond à un symbole terminal dans la section d'actions et un symbole non terminal dans la section des sauts. Toutes les cellules non peuplées lors de la génération de la matrice constituent des cas

d'erreurs et elles ne doivent pas être visitées lors de l'analyse syntaxique. Il existe trois actions possibles :

1. Décalage («shift») : Cette action est utilisée quand un symbole valide est reçu par l'analyseur syntaxique. Ce symbole ainsi que l'état indiqué par l'action de décalage sont alors mis sur la pile (l'entrée est décalée), car l'analyseur a besoin d'accumuler plus de symboles pour pouvoir reconnaître une production. Il nous faut préciser que le sommet de la pile contient l'état courant de l'analyseur.

2. Réduction («reduce») : Cette action est utilisée quand une production est reconnue sur la pile, tous les symboles faisant partie de cette production sont alors enlevés de la pile. Ils sont remplacés par le symbole non terminal produit par cette production. C'est quand une action de réduction est exécutée que la partie de la matrice qui contient les sauts est utilisée afin de déterminer dans quel état l'analyseur doit se rendre. Le symbole non terminal produit par la production (associée à l'action de réduction) ainsi que l'état se trouvant sur la pile sont utilisés pour trouver la bonne entrée dans la section des sauts.

3. Acceptation («accept») : Lorsque, nous rencontrons une action d'acceptation à la réception d'un symbole l'analyse est terminée avec succès. Remarquons que l'acceptation est, en fait, une réduction pour laquelle aucune action sémantique n'est effectuée. Remarquons aussi que cette réduction absorbe directement le symbole reçu sans le mettre sur la pile.

4.3.5.2 Génération de la matrice

La génération de la matrice suit les techniques décrites dans [24] et [26]. Dans la plupart des exemples présentés dans la littérature, la fin de séquence est indiquée par un

symbole fictif qui ne fait pas partie de la grammaire initiale. Généralement, il s'agit de la fin du fichier d'entrée. Les algorithmes de génération des matrices LR(0) ajoutent une production supplémentaire qui englobe la production de départ et qui se termine par le symbole fictif de fin de séquence. Par production de départ nous entendons la ou les productions qui génèrent le symbole de départ. Une grammaire ainsi modifiée est appelée une grammaire augmentée. Durant l'analyse syntaxique, lorsque ce symbole apparaît à l'entrée et que l'analyseur se trouve dans un état adéquat (il n'y a pas eu d'erreur dans la séquence) la réduction par la production de départ est déclenchée (directement ou indirectement). Cette réduction est suivie par l'action d'acceptation. À cette étape tous les symboles ont été consommés et toutes les réductions requises ont été exécutées. Dans notre cas, il est impossible de générer un tel symbole au niveau de l'entrée puisqu'il n'existe aucune façon de se rendre compte qu'une instance d'acquisition a terminé son activité (e.g. un terminal mobile a terminé sa connexion). Nous pouvons envisager deux approches pour résoudre ce problème.

La première approche consiste à détecter la terminaison au niveau de l'analyse syntaxique. Elle peut être implantée de deux manières.

- La première manière consiste à assumer que la grammaire tel que spécifiée initialement constitue la grammaire augmentée. Donc le symbole terminal qui se trouve à la fin de la production de départ constitue le symbole indiquant la fin de l'entrée. Dans ce cas il faut que l'action d'acceptation soit confondue avec la réduction de la production de départ.

- La deuxième manière consiste à procéder tel que décrit dans la littérature en augmentant la grammaire. Lors de l'analyse syntaxique à l'arrivée du dernier symbole terminal de la production de départ il suffit générer le symbole fictif de fin d'entrée.

N'oublions pas non plus, qu'il est possible qu'il y ait plus qu'une production de départ ou que celle-ci possède d'une alternative. Dans cette situation il faut que les symboles terminaux terminant chacune de ces alternatives soient considérés comme des symboles indiquant la fin. Malheureusement, cette technique peut uniquement être utilisée lorsque la production de départ se termine par un symbole terminal ou qu'il est possible de dériver un terminal indiquant la fin. Il est impossible de l'utiliser, par exemple, lorsque la production de départ se termine par un symbole non terminal qui résulte de la construction d'une liste. Dans ce cas, nous ne pouvons pas décider au niveau syntaxique lorsque la fin est atteinte.

La deuxième approche consiste à détecter la terminaison au niveau de l'analyse sémantique. Cette approche est beaucoup plus fiable car le contenu des messages indique toujours la fin d'une connexion. Sinon, le MSC ne serait pas capable de déconnecter les terminaux adéquatement. Dans le cadre de cette approche nous avons qu'à passer tous les symboles à l'objet «ObjetSémantique» avant d'appliquer l'algorithme d'analyse syntaxique. L'objet «ObjetSémantique» doit alors examiner les symboles reçus. À ce moment là, ou bien, durant l'exécution d'une action sémantique (quand une réduction est lancée) l'objet «ObjetSémantique» peut changer l'état d'un fanion de contrôle afin d'indiquer qu'il faut générer un symbole indiquant la terminaison. L'analyseur syntaxique va alors traiter le symbole reçu puis il va générer un symbole fictif indiquant la terminaison. Du point de vue implantation les choses sont très simples avec cette approche car la génération de la matrice et le fonctionnement de l'algorithme de la manière décrite dans la littérature. Cette approche est très similaire à la deuxième variante de la première approche, hormis que la décision est prise par la partie sémantique de l'analyse plutôt que la partie syntaxique. Donc elle permet d'offrir plus de flexibilité.

Dans l'avenir nous voulons offrir les deux possibilités aux usagers du système. Cependant, dans le cadre de ce projet nous allons implanter uniquement la deuxième

option. Finalement, outre le problème du symbole indiquant la fin de l'entrée, notre implantation suit fidèlement les méthodes présentées dans la littérature [24, 26].

4.3.5.3 Structure des classes d'implantation

La figure 4.9 présente la hiérarchie de classes en UML qui constituent l'implantation de l'analyseur syntaxique LR(0). Cette hiérarchie complète celle qui a été présentée sur la figure 3.8 dans le chapitre 3.

Dans ce projet, la plate-forme d'acquisition doit analyser plusieurs séquences d'événements à la fois. Pour cette raison, il nous faut créer un contexte d'acquisition pour chaque séquence analysée. Nous avons décidé de créer deux classes, une appelée «ClientLR» qui contient le contexte d'analyse pour chaque séquence et l'autre appelée «ServeurLR» qui contient l'algorithme principal ainsi qu'un objet «MatriceActionSaut». Chaque «ClientLR» contient un objet «Pile» ainsi qu'un objet «ObjetSémantique» qui constituent le contexte d'analyse syntaxique. La classe «ObjetSémantique» possède les méthodes qui effectuent l'analyse sémantique, c'est-à-dire que lorsque l'analyse atteint une réduction cet objet va être sollicité avec la séquence d'événements faisant partie de cette production ainsi qu'une identification (numéro) de la production. Finalement, la classe «ClientLR» contient une référence au «ServeurLR» qui lui permet d'exécuter l'analyse syntaxique. Le rôle de la classe «UsineClientLR» consiste à créer et configurer les objets «ClientLR» quand une nouvelle séquence doit être analysée. Lors de l'analyse syntaxique, chaque fois qu'un symbole terminal est reçu, l'objet «ClientLR» fait appel à l'objet «ServeurLR». Il lui passe le symbole reçu, la pile et l'objet «ObjetSémantique». L'objet «ServeurLR» exécute les actions requises et retourne les objets «Pile» et «ObjetSémantique» mis à jour. Les autres classes ainsi que leur fonctionnement seront présentées dans la section 4.3.8.

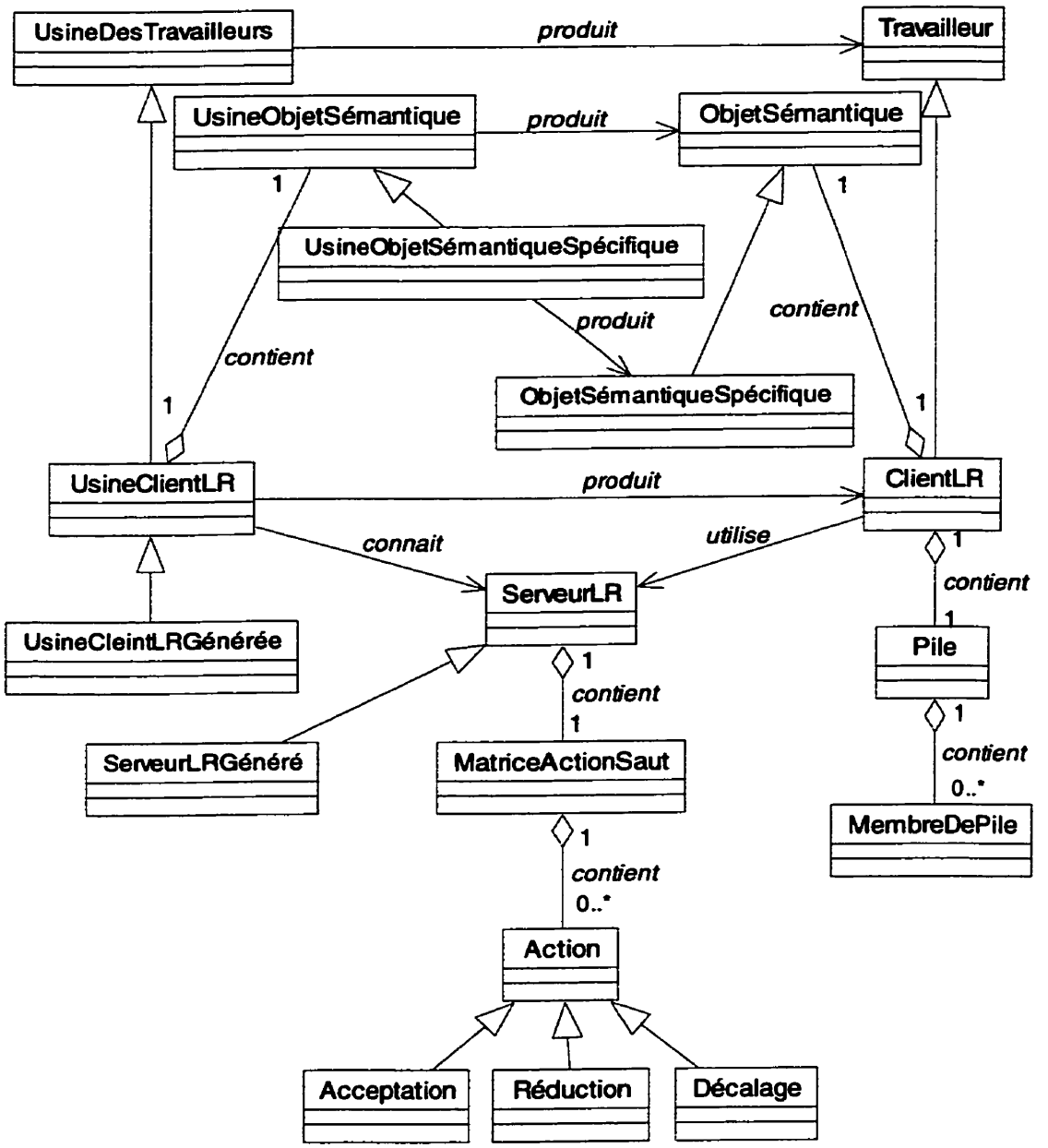

Figure 4.9 Hiérarchie des classe de l'analyseur syntaxique

4.3.6 Interaction entre le langage DDL et le langage ADL

Tel que montré à la figure 4.10 l'interaction entre le DDL et l'ADL se fait par la table des symboles. Le générateur automatique de code qui génère des parties du décodeur génère aussi une table de symboles. Cette table est utilisée par l'ADL pour faire la distinction entre les symboles terminaux et les symboles non terminaux. Elle permet aussi de faire la relation entre les noms abstraits utilisés dans les spécifications et leur identification réelle contenue dans les en-têtes.

Enfin, nous pouvons remarquer que la plate-forme conçue dans le cadre de ce projet fonctionne selon un principe similaire aux outils LEX et YACC [33]. Dans les deux cas, les systèmes sont composés de deux parties. Une partie qui extrait les symboles terminaux (LEX, dans notre cas DDL) et une partie qui effectue l'analyse syntaxique (YACC, dans notre cas ADL). Les deux systèmes offrent aussi une interface de programmation au niveau des actions sémantiques. Les seules différences sont le fait que LEX et YACC implantent la méthode LALR («look ahead LR») qui correspond à la méthode LR(1), optimisée et que notre système permet d'effectuer plusieurs analyses syntaxiques à la fois.

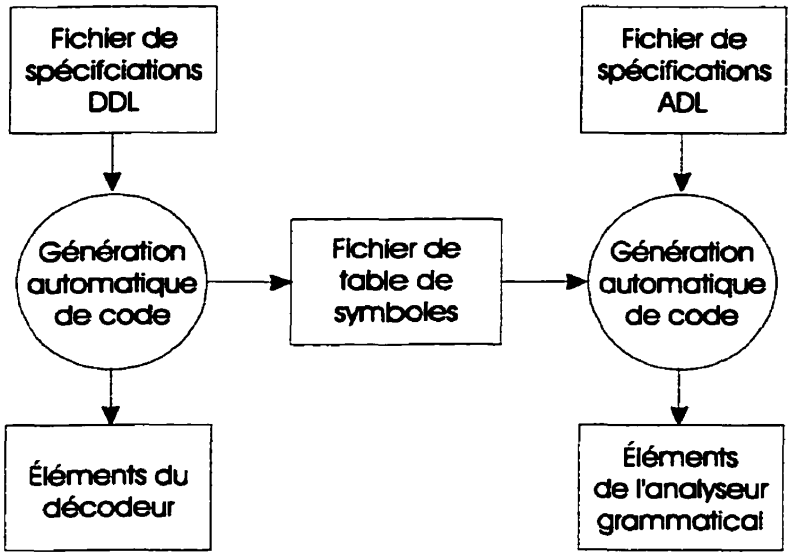

Figure 4.10 Interaction entre le DDL et l'ADL

4.3.7 Le langage ADL

Le langage ADL permet de définir les séquences d'événements qui seront reçus à partir du MSC. En effet, il s'agit de la définition d'une grammaire LR(0). Pour cette raison, nous nous sommes inspirés du format de notation de Backus-Naur [25, 26, 27, 34, 35] que nous avons présenté plus tôt. Cette notation est utilisée par les outils tels que YACC [33]. Il est important de noter qu'avec notre approche chaque production doit être identifiée. C'est pourquoi nous ne permettons pas d'utiliser l'opérateur «|» qui indique des séquences de symboles alternatives pour une même production. Au lieu d'utiliser cet opérateur il faut que l'usager définisse une autre production produisant le même terminal.

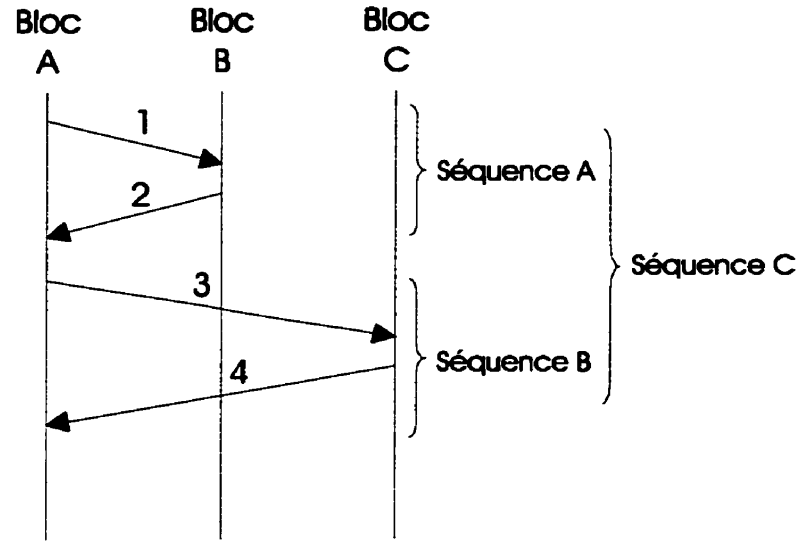

Figure 4.11 Séquence d'événements entre trois blocs du logiciel de contrôle

```
preamble {
  semantic = UsineObjetSémantiqueSpécifique;
  end_of_input = semantic;
}

grammar {
  1: C := A B;
  2: A := 1 2;
  3: B := 3 4;
}
```

Figure 4.12 Exemple des spécifications en ADL

La figure 4.11 présente un exemple fictif de séquence d'événements entre trois blocs du logiciel de contrôle du MSC. La figure 4.12 présente la description en ADL de cette séquence. On peut y distinguer deux sections principales :

1) La section «preamble» contient les paramètres de l'interface de programmation. Elle possède les deux variables suivantes :
   a) «semantic», indique le nom de la classe qui crée les objets sémantiques spécifiques. Ces objets sémantiques implantent l'analyse sémantique.

b) «end_of_input», indique quel type de fin de séquence doit être utilisé. Le mot réservé «terminal» indique que la première méthode présentée dans la section 4.3.5.2 va être utilisée (détection au niveau de l'analyse syntaxique), tandis que le mot réservé «semantic» indique que c'est la deuxième méthode qui va être utilisée (détection au niveau de l'analyse sémantique). Actuellement, seule la deuxième méthode est implantée.

2) La section «grammar» contient la définition de la grammaire que les séquences doivent respecter. Elle est composée de la définition des productions. La première production est la production de départ. Chaque production est définie comme suit :
   a) Numéro de production : Les numéros doivent être uniques et la numérotation commence par 1. Ce numéro est passé à l'objet sémantique quand une production est exécutée.
   b) Nom du symbole non terminal qui est produit par la production.
   c) Liste des symboles qui font partie de la production.

Le tableau 4.2 présente la grammaire du ADL décrite tout comme la grammaire du DDL à l'aide de la notation Backus-Naur. Nous respectons la même convention que dans notre description du DDL.

Tableau 4.2 Grammaire du langage ADL

No. de production	Production en format BNF	Signification sémantique
1	Adl ::= Preamble Grammar	Définition globale du langage.
2	Preamble ::= **preamble** { SemFac_specs EndOfInput_specs }	La section «preamble» contient deux éléments placés dans n'importe quel ordre.
	| **preamble** { EndOfInput_specs SemFac_specs }	

Tableau 4.2 (suite)

No. de production	Production en format BNF	Signification sémantique
3	EndOfInput_specs ::= **end_of_input = semantic;**	Actuellement, la section qui spécifie le comportement à la fin de l'entrée permet uniquement de spécifier la valeur «**semantic**»
4	SemFac_specs ::= **semantic** = STRING ;	Le nom de la classe qui va générer l'objet sémantique peut être n'importe quelle séquence de caractères. Remarquons que ce nom doit respecter la norme C/C++. Cette classe est compilée séparément donc nous n'avons pas à vérifier la validité du nom de la classe.
5	Grammar ::= **grammar** { ProductionList_spec }	La grammaire est constituée d'une liste de productions.
6	ProductionList_spec ::= ProductionList_spec Production | Production	Définition de la liste des productions, notons que l'unicité de chaque production tant par son numéro d'identification que par sa structure est effectuée au niveau sémantique.
7	Production ::= UNSIGNED : STRING := ElementList_spec;	Chaque production est composée d'un numéro d'identification. Notons qu'une production vide doit contenir un seul élément nul (voir production 9).

Tableau 4.2 (suite)

No. de production	Production en format BNF	Signification sémantique
8	ElementList_spec ::= ElementList_spec Element | Element	Liste des éléments terminaux et non terminaux, faisant partie de chaque production.
9	Element ::= STRING | **null**	Chaque élément est défini par une chaîne de caractères ou est assigné à nul (élément vide). La distinction entre un symbole terminal et un symbole non terminal est effectuée au niveau sémantique.

4.3.8 Génération automatique de code et interface de programmations

Si nous revenons à la figure 4.9, nous pouvons remarquer que nous avons utilisé la même approche pour la génération du code que celle utilisée dans le cas du DDL, c'est-à-dire que deux classes contenant les méthodes de configuration sont générées : «UsineClientLRGénérée» et «ServeurLRGénéré». La configuration consiste à créer, initialiser et interconnecter les instances des classes effectuant à l'analyse. Il faut noter que les classes «ObjetSémantiqueSpécifique» et «UsineObjetSémantiqueSpécifique» ne sont pas générées, mais leur implantation est liée lors de l'édition des liens pendant la création du programme exécutable.

D'autre part, les classes «ObjetSémantique» et «UsineObjetSémantique» constituent uniquement l'interface de programmation. La véritable implantation de l'analyse sémantique est effectuée par les classes qui en héritent. À la figure 4.9, il s'agit de la classe «ObjetSémantiqueSpécifique». Elle est créée par la classe «UsineObjetSémantiqueSpécifique», et hérite à son tour de la classe

«UsineObjetsSéamantique». Les classes «ObjetSémantiqueSpécifique» et «UsineObjetSémantiqueSpécifique» constituent la partie implantée par l'usager du système. Tel que nous l'avons dit plus tôt, la classe «ObjetSémantiqueSpécifique» sera sollicitée avec le numéro de production et une séquence de terminaux lorsqu'une réduction sera exécutée. La méthode membre de la classe «ObjetSémantiqueSpécifique» qui sert d'entrée à l'analyse sémantique peut retourner un objet qui hérite de la classe «MembreDePile». Cet objet correspond au symbole résultant de la réduction. Il est empilé par l'analyseur syntaxique pour pouvoir être réutilisé plus tard par l'analyseur sémantique.

CHAPITRE V – ÉVALUATION DES PERFORMANCES

Dans ce chapitre nous allons nous pencher sur les performances de la plate-forme d'acquisition. Nous allons commencer par présenter un aperçu théorique des performances des algorithmes impliqués dans les différentes parties. Ensuite, nous allons montrer les résultats des essais de la plate-forme avec les données réelles acquises lors de l'expérience préliminaire. Remarquons que notre objectif n'est pas d'analyser les performances d'une application en particulier. Nous voulons seulement vérifier que notre plate-forme peut servir de base pour des applications d'analyse en temps «presque» réel.

5.1 Performances théoriques

L'objectif de cette partie est de vérifier si la contrainte 4 «Traitement en temps «presque» réel» peut être satisfaite. Rappelons, que dans notre contexte, le traitement en temps «presque» réel veut dire que les événements doivent traverser notre plate-forme à partir de leur bloc de génération jusqu'à l'application qui exécute les actions sémantiques dans des délais raisonnables. De manière arbitraire, nous considérons les délais de quelques secondes comme raisonnables. D'autre part, le traitement en temps «presque» réel implique que les délais ne doivent pas varier de façon significative, selon les heures de la journée. En effet, cela correspond au fait que la relation entre les taux d'arrivées et de service doit respecter la contrainte suivante : taux d'arrivées / taux de service < 1. Arnold et al [55] présentent une approche qui permet de dimensionner les différents éléments d'un système de mesure fonctionnant en temps réel. Dans notre cas, il s'agit uniquement d'évaluer les performances de notre approche qui, tel que décrit dans le chapitre 2, sont

limitées par l'utilisation d'un ordinateur personnel simple (par opposition à une machine dédiée).

Du point de vue théorique, nous devons nous assurer que les approches prises dans les différentes parties de notre système garantissent un traitement constant en temps et espace pour chaque événement reçu. Le temps de traitement de chaque événement pris individuellement doit être indépendant du nombre de séquences d'événements en reconstruction et du nombre total des événements contenus dans le système.

La figure 5.1 résume les étapes majeures du traitement des données avec notre plate-forme. Le tableau 5.1, quant à lui, nous donne un sommaire des justifications des performances constantes (O(k)) en temps d'exécution et espaces pour chacune de ces étapes pour les événements pris individuellement.

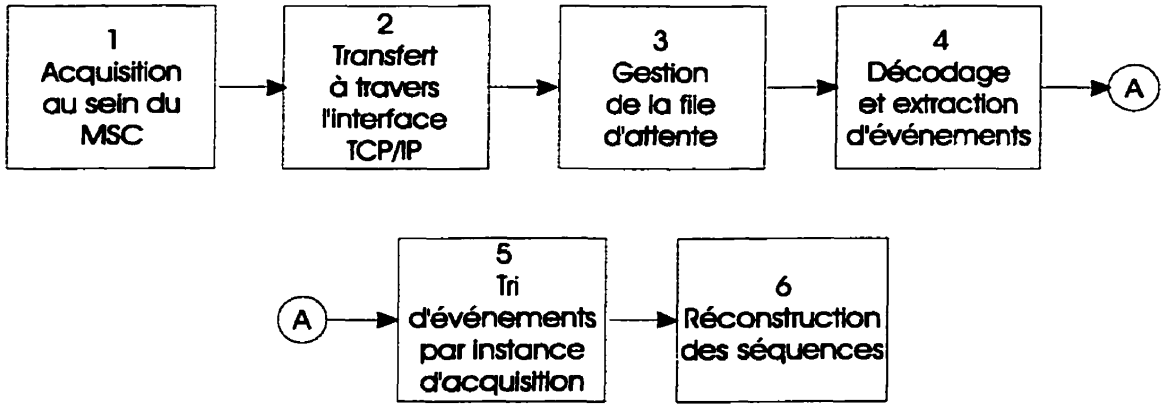

Figure 5.1 Résumé d'étapes majeures du traitement des données

Tableau 5.1 Sommaire des justifications des performances constantes

Section	Justification des performances O(k) en temps d'exécution et espace.
1. Acquisition au sein du MSC	La taille de l'espace tampon de mémoire est constante et le remplissage de l'espace tampon est limité par le temps.
2. Transferts à travers l'interface TCP/IP	Grâce à l'approche de l'étape 1 la taille des espaces tampons transférés peut être considérée comme constante donc le temps du transfert doit être à peu près constant puisqu'il est impossible d'avoir des collisions.
3. Gestion de la file d'attente	Le poste de travail est exclusivement utilisé comme plate-forme d'acquisition c'est pourquoi le disque n'est utilisé de façon notable que par notre application. Les taux de transfert des disques sont de l'ordre de plusieurs milliers d'octets par seconde [51]. Or les taux d'arrivées des données sont de l'ordre de dizaines d'octets par seconde (chapitre 3). D'autre part, les parties qui effectuent l'écriture et la lecture sur le disque maintiennent les fichiers ouverts donc les structures sont déjà contenues en mémoire [50]. Enfin, précisons que les temps de recherche moyen type sur un disque dépend surtout de ses spécification physiques (e.g. géométrie, vitesse de rotation, etc.). Il est tout à fait indépendant de la quantité de données stockées quand le taux de fragmentation est bas [51]. C'est pourquoi nous pouvons assumer que les temps d'écriture et lecture sont constants et prennent un espace constant.
4. Décodage et extraction des événements	Le temps de décodage d'événements dépend du nombre de champs contenus dans les événements. La structure des événements est constante pour chaque type d'événement. D'autre part, les décodeurs sont associés en utilisant les tables de dispersion ce qui garantit l'exécution en temps constant. L'espace mémoire pris par les événements décodés sera pris en considération dans les parties ultérieures.

Tableau 5.1 (suite)

5. Tri d'événements par instance d'acquisition	La structure de données qui permet de stocker les travailleurs effectuant la reconstruction des séquences est une table de dispersion. Les temps d'accès sont constants et la mémoire prise par chaque travailleur (ou objet «ClientLR»), mis à part la part la pile, est constante.
6. Reconstruction des séquences	L'algorithme LR(0) s'exécute en temps et espace linéaire ($O(n)$) pour une séquence d'entrée. C'est pourquoi, du point de vue d'un événement pris individuellement, le temps pris pour son traitement et l'espace pris pour son stockage (pile contenue dans les travailleurs ou objets «ClientLR») peuvent être considéré comme indépendants des autres événements donc constants.

5.2 Performances expérimentales

De manière pratique, il s'agit de vérifier si les deux contraintes suivantes sont respectées :

1. Le temps de passage des éléments à travers le système ne dépasse pas quelques secondes.
2. Il n'y a pas de variation significative de délais durant les différentes heures de la journée.

Pour des raisons de logistique il est impossible de connecter notre plate-forme à un système réel. Cependant, lors de l'expérience préliminaire nous avons acquis plus de 24 heures d'événements générés par un commutateur numérique en région urbaine. Ces données ont la forme, en effet, de la file d'attente de messages, qui se trouve entre l'application qui reçoit les données et l'application qui effectue le traitement. Lors de cette acquisition, outre les données proprement dites nous avons aussi créé un ficher de

statistiques qui indique l'heure exacte du début de l'acquisition ainsi que les temps relatifs de réception de chaque groupe d'événements. Nous allons pouvoir exploiter ces données afin de mesurer les capacités pratiques de notre plate-forme. D'autre part, lors de cette même expérience, nous avons prouvé que l'implantation au niveau du MSC est en mesure de satisfaire la contrainte 4 («Traitement en temps «presque» réel»). Nous avons aussi montré que l'application qui est responsable de recevoir et stocker les données ne génère pratiquement pas d'activité sur la plate-forme d'acquisition. Donc s'il y a un goulet d'étranglement il se trouve au niveau du traitement. C'est pourquoi nous nous proposons d'effectuer des expériences qui vont se baser sur les données acquises afin de tester l'application qui effectue le traitement des données.

5.2.1 Configuration expérimentale

Tel que nous avons remarqué lors de l'expérience préliminaire, l'application qui effectuait le stockage ne prenait qu'une partie très faible du temps de l'exécution de l'UCT (inférieur à 1%). C'est pourquoi nous pouvons considérer que les expériences avec la configuration présentée à la figure 5.2 seront suffisamment précises. Cette configuration inclut uniquement la partie qui effectue le traitement de données. Du point de vue pratique, la plate-forme expérimentale est constituée par un ordinateur personnel basé sur un processeur Pentium II de Intel fonctionnant à une vitesse d'horloge de 233MHz avec 128Mo de mémoire. Afin de tester les performances, nous allons prendre des mesures aux trois endroits indiqués par des numéros sur la figure 5.2. Ces mesures vont constituer trois expériences distinctes.

La première expérience va consister à analyser le comportement du décodeur fonctionnant tout seul, c'est-à-dire que nous allons procéder au décodage de tous les événements contenus dans les groupes acquis.

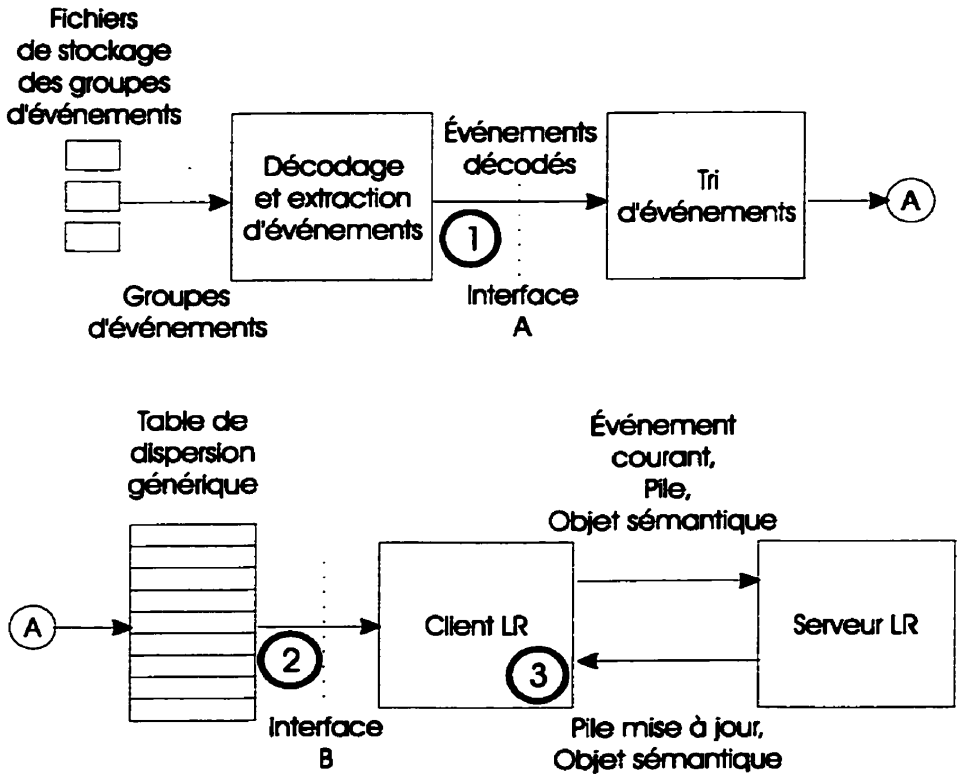

Figure 5.2 Mesures de délais

La deuxième expérience va consister à analyser le comportement des méthodes du démarrage et de l'arrêt des travailleurs participants au tri des événements, c'est-à-dire nous allons procéder au décodage et au tri de tous les éléments reçus. Le tri va se faire selon les numéros d'identité de chaque événement (e.g. numéro du téléphone, numéro de cellule, etc.). Cependant, puisque nous n'allons pas reconstruire les séquences d'événements nous allons démarrer un travailleur pour chaque message reçu, ensuite nous allons consommer ce message et finalement nous allons arrêter le travailleur. De cette manière, nous allons simuler les activités les plus exigeantes en terme de temps d'exécution dans la partie qui effectue le tri de messages. Il faut noter que dans cette expérience nous n'allons pas vérifier la qualité des méthodes de dispersion.

La troisième expérience va consister à analyser le comportement des méthodes LR(0) permettant la reconstruction des séquences. Nous allons décoder uniquement les messages pouvant faire partie des séquences. Afin de générer l'objet «Serveur LR» nous allons définir une grammaire qui va inclure les listes de messages les plus fréquemment rencontrés. Dans le cas du mode numérique ces messages vont inclure les messages périodiques présentant la qualité de la réception radio [16] («Radio Quality Message»). Dans le cas du mode analogique ces messages vont inclure les requêtes de changement de cellule [18] («Analogue Handoff Request»). De cette manière, nous allons nous assurer que le système sera chargé au maximum. En effet cette expérience va être équivalente à un des pires cas d'analyse. Cette expérience va nous permettre de vérifier la performance de la table de dispersion et la performance de notre implantation de l'algorithme d'analyse grammaticale LR(0).

5.2.2 Méthode de mesures

Lors de nos expériences nous allons exploiter les données qui ont été acquises durant l'expérience préliminaire, c'est-à-dire que nous allons instrumenter le code en y ajoutant une classe spéciale d'analyse qui va prendre l'heure réelle lors de la lecture d'un groupe de messages. Ensuite, lors de la lecture du groupe suivant, l'heure va être prise à nouveau. La différence entre ces deux mesures sera comparée avec la différence des temps relatifs enregistrés lors de l'acquisition (durant l'expérience préliminaire). Nous avons deux possibilités. La première possibilité, est que notre temps d'exécution arrive en retard par rapport au temps enregistré lors de l'expérience préliminaire. La deuxième possibilité telle qu'indiquée à la figure 5.3 est que notre exécution se termine plus tôt que le temps enregistré lors de l'expérience préliminaire. Dans ce cas nous sommes en avance et il nous faut comptabiliser tous les temps d'avance afin de les ajouter à toutes les lectures subséquentes de l'horloge réelle. Cette comptabilité est requise car, autrement, nos

mesures prendraient en compte les avances que nous pouvons gagner lors des heures de bas trafic ce qui masquerait les délais éventuels durant les heures de pointe.

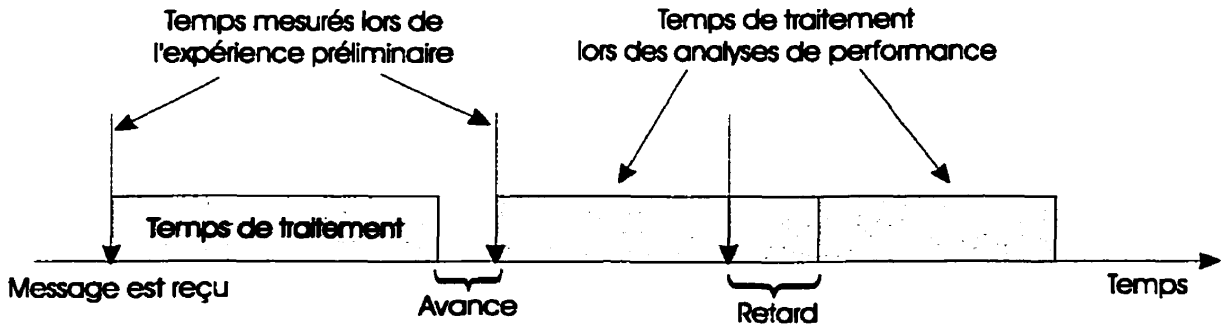

Figure 5.3 Représentation des avances et retards

Lors de la troisième expérience, puisque nous n'utilisons pas toutes ces données, nous enlèverons les temps d'accès au disque du temps de traitement. Nous devons réaliser ceci, car les groupes d'événements contiendront toujours tous les événements possibles. Donc le temps de traitement des événements d'intérêt serait faussé par le temps de chargement des autres événements. Ceci est particulièrement sensible dans le cas où la proportion des événements d'intérêt baisserait. Nous verrions alors une fausse augmentation de leur temps de traitement. C'est d'ailleurs pour cette même raison que lors de la troisième expérience nous allons uniquement décoder les événements d'intérêt.

Dans nos analyses nous sommes essentiellement intéressés à mettre en évidence les retards durant le cycle de 24 heures, la valeur du temps moyen de traitement de chaque événement, et finalement la variation de cette valeur sur une période de 24 heures. Notre analyse théorique indique que ce temps devrait être pratiquement constant. Ces mesures vont nous permettre de voir si notre système est capable de fournir des résultats en temps «presque» réel.

5.2.3 Résultats

Nous allons maintenant présenter les résultats de chacune des expériences. Afin de réduire la quantité de données, nous avons effectué des moyennes par tranches de 10 minutes, c'est-à-dire que, à part pour les valeurs maximales, nous allons baser nos calculs sur des valeurs moyennes de période de 10 minutes. Notons aussi que l'acquisition a débuté vers 9h30 et s'est terminée vers 10h30. Dans le cadre de nos expériences nous avons arrêté l'analyse un peu avant 9h30 à cause de contraintes d'espace disque disponible. C'est pourquoi, sur la plupart des figures nous pouvons remarquer un manque de continuité lors de cette période.

5.2.3.1 Résultats de la première expérience

La figure 5.4 nous montre la distribution des arrivées de tous les événements lors des 24 heures d'acquisition. La figure 5.5 nous montre le taux d'utilisation de la plate-forme. Le taux d'utilisation est calculé en divisant la durée pendant laquelle nous traitons les événements (temps actif) d'un groupe par le temps disponible entre la réception des groupes consécutifs (temps total). Nous pouvons y remarquer que celui-ci ne dépasse pas les 5%. Le tableau 5.2 résume les autres résultats. Nous pouvons remarquer que le traitement des données est à peu près constant pour toutes les tranches de 10 minutes puisque l'écart type est de l'ordre de 2% par rapport à la moyenne. D'autre part, il faut noter que la plate-forme est constituée par un système UNIX (Linux) donc tous les processus marchent à temps partagé. Il se peut donc que des tâches périodiques ou des tâches d'arrière plan influencent l'activité de notre système. Il faut aussi préciser que les délais qui sont survenus lors de l'expérience concernent des groupes d'événements individuels et sont très peu fréquents. En analysant la situation, nous avons remarqué que dans le fichier décrivant les séquences de groupes d'événements reçus, il y a des périodes relativement longues avant la réception de certains groupes. Chacun de ces groupes est

toujours suivi par la réception d'un autre groupe dans un délai extrêmement court. C'est toujours ce groupe suivant qui cause les délais que nous remarquons. Ce comportement est dû au fonctionnement des éléments qui précèdent notre application de réception de données. L'élément fautif peut être au niveau du MSC, tout aussi bien qu'au niveau du poste de travail. La figure 5.6 présente la distribution de la proportion des groupes en retard par rapport à tous les groupes reçus. On peut, remarquer que cette proportion suit la densité du trafic. Enfin, notons que dans cette expérience nous avons inclus les lectures à partir du disque. Donc cette expérience nous prouve aussi que l'opération du disque se fait en temps constant.

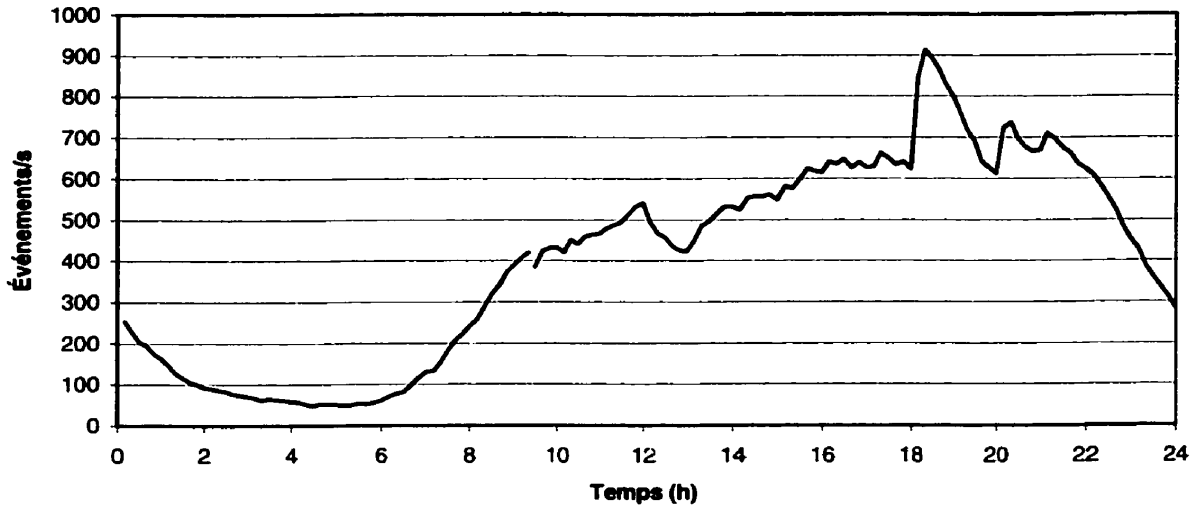

Figure 5.4 Expérience 1, distribution du taux d'arrivée des événements

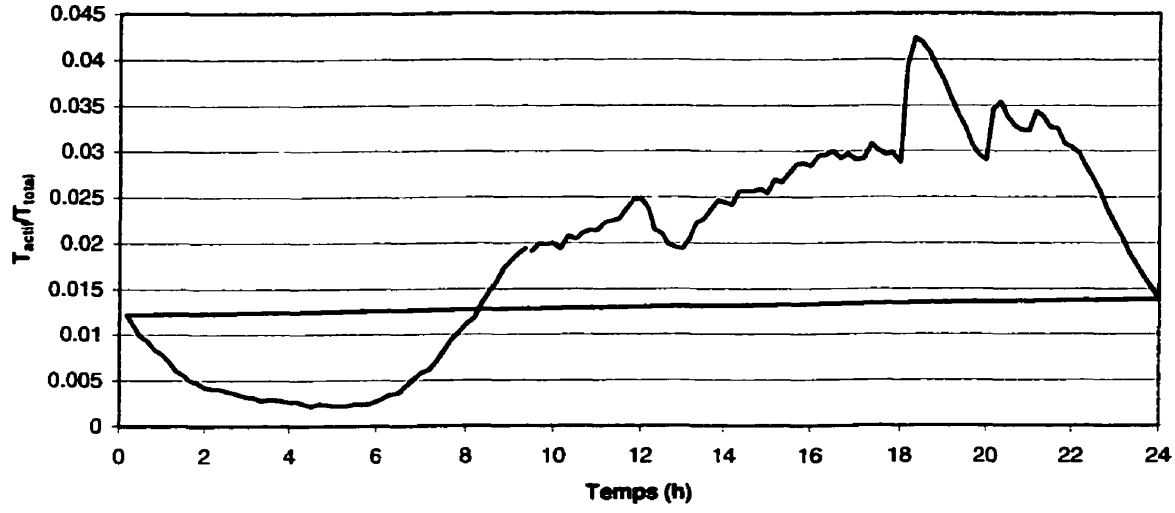

Figure 5.5 Expérience 1, distribution du taux d'utilisation de la plate-forme

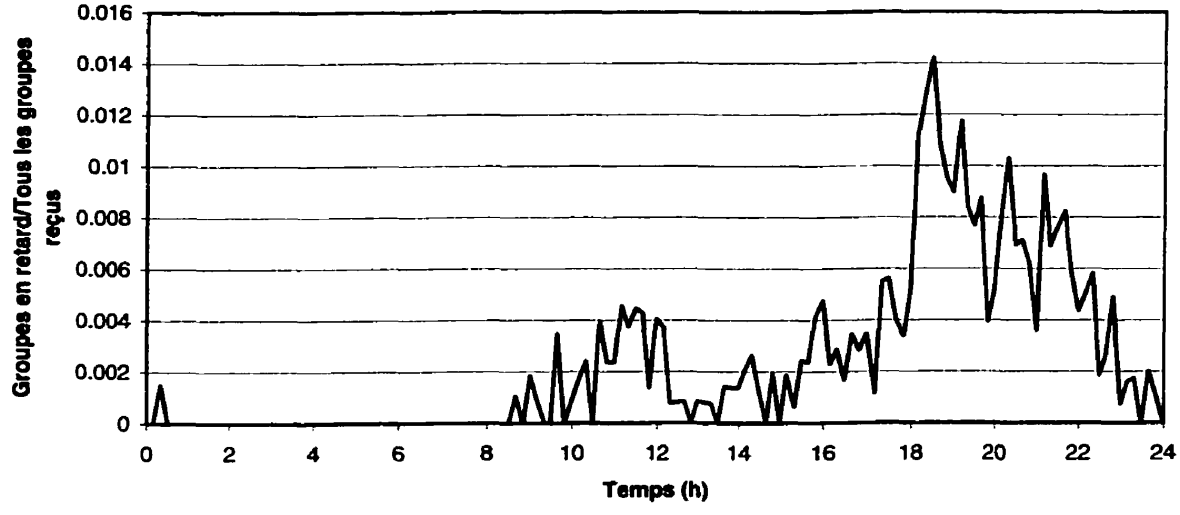

Figure 5.6 Expérience 1, distribution de la proportion des groupes en retard

Tableau 5.2 Résultats de l'expérience 1

Statistiques par tranches de 10 minutes	Valeur	Remarques et explications
Temps de traitement moyen par événement	47µs	Nous connaissons seulement les temps des arrivées de différents groupes d'événements, nous ne connaissons pas les temps de génération des événements en particulier. C'est pourquoi nous pouvons uniquement calculer les temps moyens de traitement sur les tranches de 10 minutes. Le temps de traitement comprend le temps de lecture d'un groupe de donnée, à partir du disque et le décodage de ces données.
Écart type du temps de traitement moyen	1.32µs	
Valeur maximale du temps de traitement moyen	50µs	
Valeur minimale du temps de traitement moyen	44µs	
Valeur moyenne du retard par groupe d'événements	1.28ms	Lors du fonctionnement de la plate-forme certains groupes d'événements ont été traités en retard. Nous pouvons remarquer que l'écart type est plus élevé que la moyenne. Ceci est dû au fait que le nombre de groupes d'événements ayant subi un retard est très faible (d'après les statistiques qui suivent).
Écart type du retard par groupe d'événements	1.99ms	
Valeur maximale du retard	7.22ms	
Pourcentage moyen des groupes qui ont été traités en retard	0,24%	Sur toutes les tranches de 10 minutes, en moyenne 0.24% des groupes d'événements ont été traités en retard.
Nombre maximal de groupes consécutifs ayant subi un retard	1	Pendant toute l'expérience nous avons seulement observé des retards de groupes individuels. Il n'y a eu aucune occurrence de retard de plus qu'un groupe à la fois.

5.2.3.2 Résultats de la deuxième expérience

Lors de la deuxième expérience nous avons remarqué un comportement à peu près identique à celui de la première expérience. Nous voyons que les durées d'exécution moyennes sont sensiblement supérieures à celles de l'expérience précédente. La distribution des arrivées des événements était identique. Le tableau 5.3 résume ces résultats et la figure 5.7 donne la distribution de la charge de la plate-forme.

Tableau 5.3 Résultats de l'expérience 2

Statistiques par tranches de 10 minutes	Valeur
Temps de traitement moyen par événement	72µs
Écart type du temps de traitement moyen	1.32µs
Valeur maximale du temps de traitement moyen	77µs
Valeur minimale du temps de traitement moyen	69µs
Valeur moyenne du retard par groupe d'événements	2.25ms
Écart type du retard par groupe d'événements	3.49ms
Valeur maximale du retard par groupe d'événements.	12.97ms
Pourcentage moyen des groupes qui ont été traités en retard	0,26%
Nombre maximal de groupes consécutifs ayant subi un retard	1

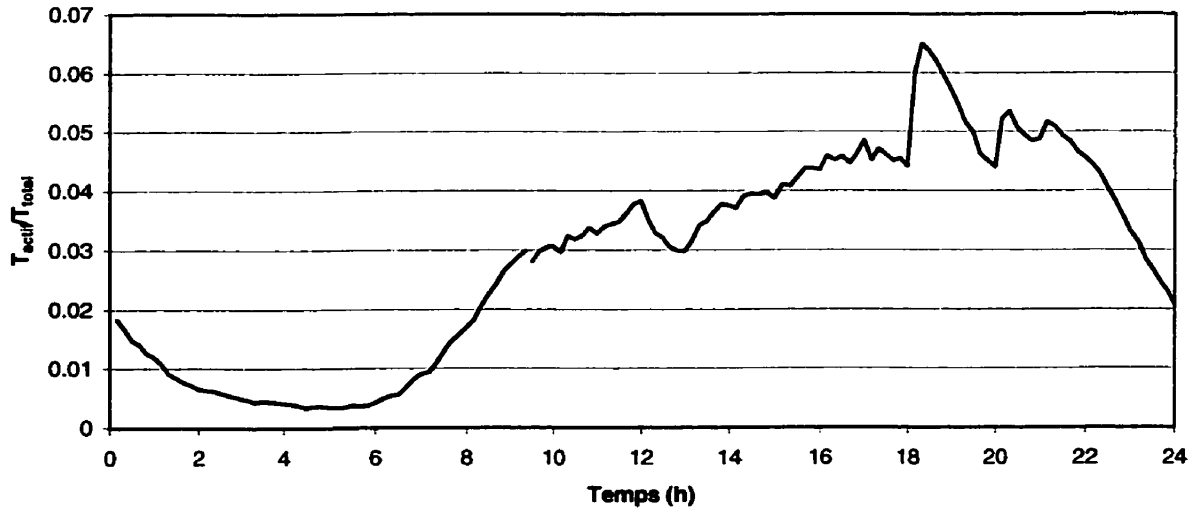

Figure 5.7 Expérience 2, distribution du taux d'utilisation de la plate-forme

5.2.3.3 Résultats de la troisième expérience

La troisième expérience démontre le fonctionnement du traitement complet des données. Tel qu'il a été dit plus tôt, nous avons utilisé seulement les quatre événements les plus fréquents du mode numérique et du mode analogique, afin de démontrer le fonctionnement global. En effet, d'après les statistiques nous avons utilisé 51% de tous les événements disponibles. La figure 5.8 présente la distribution des arrivées des événements sélectionnés. Le tableau 5.4 présente les statistiques obtenues. La figure 5.9 présente les temps de traitement par événements. Nous pouvons remarquer que les taux semblent augmenter durant la nuit ce qui peut sembler étrange. En fait ceci est uniquement dû à un changement de proportions des événements, et à une baisse générale de l'intensité du trafic. La figure 5.10 montre le changement des proportions des événements. Elle montre la proportion du taux des arrivées des événements d'intérêt versus le taux des arrivées de tous les autres événements. Ce phénomène fait en sorte qu'il y a moins d'événements par groupe donc l'activité liée aux tâches connexes devient plus importantes par rapport à celle liée à l'analyse. Cela donne l'impression que le temps de traitement par événement est plus long. Même si tel que nous l'avons expliqué plus tôt nous avons soustrait le temps de charge des blocs en mémoire et nous ne décodons pas tous les événements. Tous les groupes doivent être parcourus afin d'extraire les événements d'intérêt qui sont alors très peu fréquents.

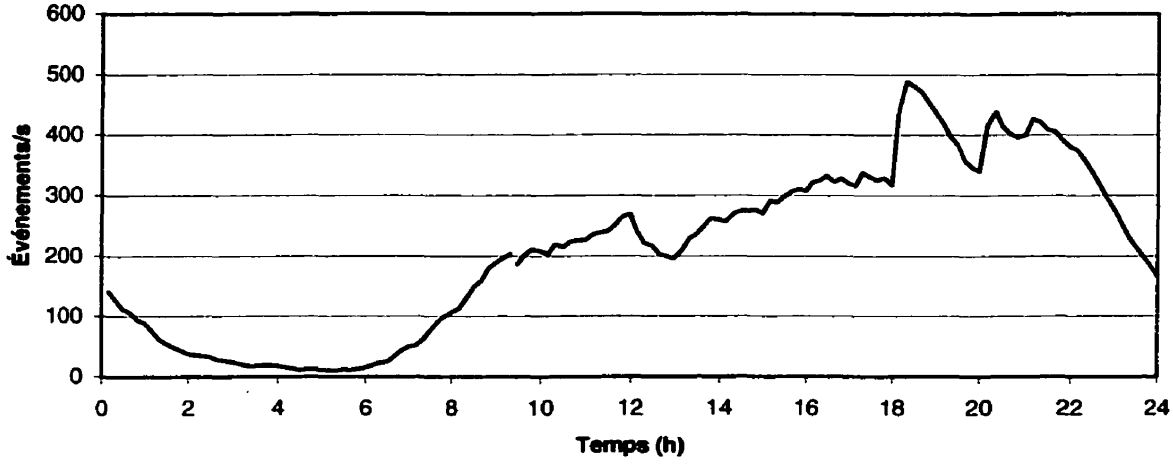

Figure 5.8 Expérience 3, distribution du taux d'arrivée des événements d'intérêt

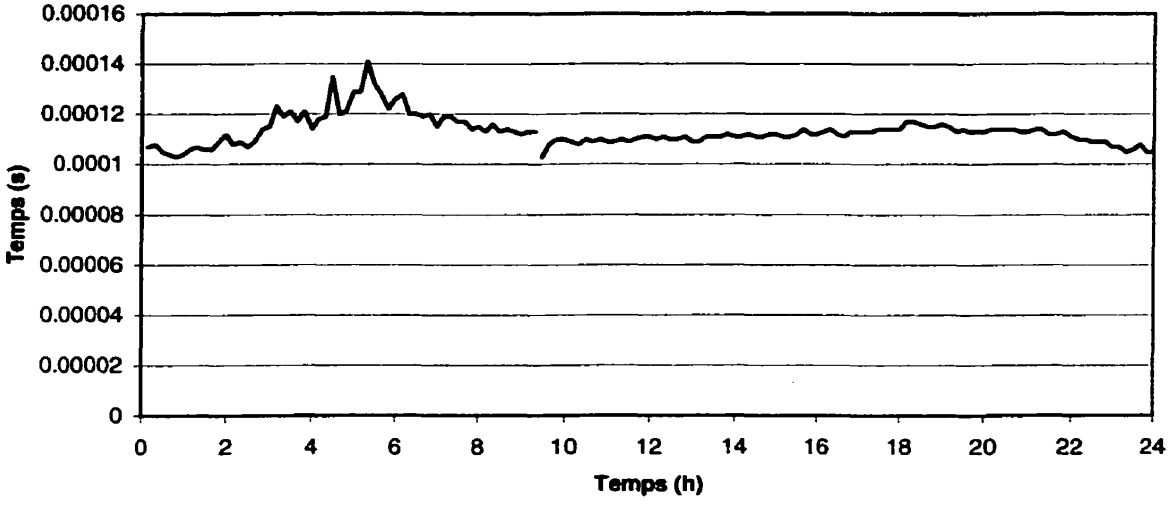

Figure 5.9 Expérience 3, distribution du temps de traitement par événement

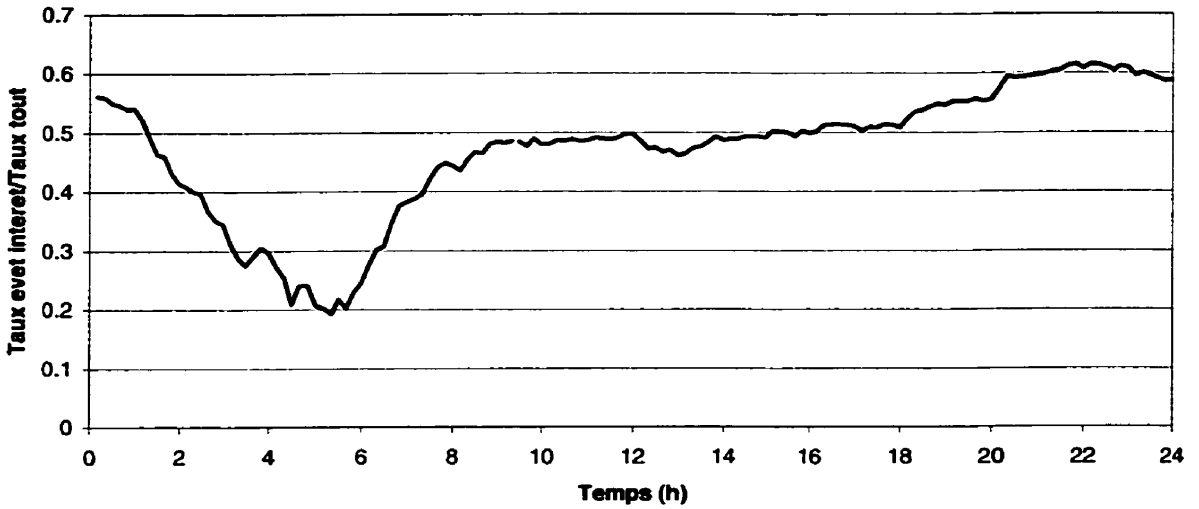

Figure 5.10 Expérience 3, distribution de la proportion des événements d'intérêt

Tableau 5.4 Résultats de l'expérience 3

Statistiques par tranches de 10 minutes	Valeur	Remarques et explications
Temps de traitement moyen par événement	113µs	. Tous les résultats sont similaires à ceux produits par les expériences précédentes.
Écart type du temps de traitement moyen	6µs	
Valeur maximale du temps de traitement moyen	141µs	
Valeur minimale du temps de traitement moyen	103µs	
Valeur moyenne du retard par groupe d'événements	1.7ms	
Écart type du retard par groupe d'événements	2.6ms	
Valeur maximale du retard par groupe d'événements	14.4ms	

Tableau 5.4 (suite)

Statistiques par tranches de 10 minutes	**Valeur**	**Remarques et explications**
Pourcentage moyen des groupes qui ont été traités en retard	0,26%	
Nombre maximal de groupes consécutifs ayant subi un retard	1	
Statistiques par tranches de 10 minutes	**Valeur**	**Remarques et explications**
Nombre moyen d'éléments par clé de la table de dispersion	1.06	Dans cette expérience il nous fallait vérifier le comportement de la table de dispersion et des fonctions génériques de dispersion. Nous avons calculé les moyennes du nombre d'événements stockés dans la table de dispersion et le nombre d'entrées utilisées dans cette table. Ces mesures semblent montrer que le mécanisme de dispersion fonctionne bien. Nous pouvons l'affirmer d'autant plus que le temps de traitement moyen et l'écart type associés indiquent une faible variation.
Écart type du nombre d'éléments par clé de la table de dispersion	0.02	

5.3 Conclusion

Après avoir effectué les trois expériences nous avons une indication que le traitement en utilisant la technique présentée peut être effectué en temps «presque» réel car le traitement n'ajoute pas de délais variables à aucune de ses étapes. Il faut tout de même remarquer que nous avons observé un comportement qui ajoute des délais sporadiques lors du transfert des données. Dans notre cas les délais engendrés dans notre système sont tout à fait minimes (valeur maximale sur 24 heures : 14.4 ms) et très peu fréquents (0.26% de groupes arrivent en retard). Nous avons remarqué, cependant, que la fréquence de ces délais dépend de la densité du trafic. C'est pourquoi, il se peut qu'à partir d'un niveau du

débit que nous n'avons pas atteint dans notre projet, ces retards puissent devenir plus importants et nuire au fonctionnement du système. D'autre part, dans nos expériences nous n'avons pas implanté d'application d'analyse. La phase suivante de ce projet serait d'implanter une série d'applications pour tester différentes possibilités d'analyses.

D'autre part, tel que nous avons expliqué dans le chapitre 3 le système ne peut pas s'effondrer à cause d'un trafic trop intense. Pendant une période ou le trafic dépasserait les capacités de la plate forme nous observerions uniquement une augmentation et une accumulation du délai (à moins que la capacité du lien TCP/IP ne soit dépassée). Cependant, il faudrait quand même tester les limites de notre plate-forme en la simulant avec des densités de trafic beaucoup plus élevés. Cela nous permettrait de déterminer la limite maximale de notre système à fournir des résultats en temps «presque» réel.

CONCLUSION

Dans le cadre de notre projet, nous avons mis au point une plate-forme permettant d'acquérir des événements asynchrones. Ces événements résultent de l'activité de l'interface radio d'un commutateur numérique AXE-10 d'Ericsson. La partie essentielle de notre travail consistait à intégrer une méthode d'analyse syntaxique du type LR(0) afin de permettre la reconstruction des séquences d'événements. Cette analyse syntaxique sert en quelque sorte à extraire les séquences qui respectent un certain nombre de règles décrites par une grammaire spécifiée par l'usager. Les séquences extraites peuvent être transmises à une application d'analyse qui effectue une interprétation de leur signification sémantique. D'autre part, dans notre conception nous avons consacré beaucoup d'efforts afin de rendre notre plate-forme aussi générique que possible. Pour ce faire, nous avons défini deux langages permettant de générer le code des parties variables de la plate-forme d'acquisition. De cette manière, il est facile de modifier la plate-forme afin de s'adapter rapidement aux changements au niveau des requis d'analyse, de l'implantation du protocole, et même du protocole lui-même.

Nous avons démontré le fonctionnement de notre plate-forme en effectuant une analyse théorique et par des tests basés sur des données réelles. Ces tests ont montré la possibilité du traitement en temps «presque» réel, tout en gardant une large marge pour l'exécution des analyses sémantiques.

En effet, notre système constitue la première application de l'analyse syntaxique dans le domaine de la supervision et analyse du fonctionnement des réseaux. C'est pourquoi nous pouvons considérer notre travail original comme une première étape dans l'établissement d'une nouvelle méthodologie d'analyse. Après cette première étape, il reste encore beaucoup de travail afin d'éprouver et mettre au point cette méthodologie. Tout

d'abord, il faudrait effectuer des tests avec des applications réelles, car n'oublions pas que nous avons effectué des tests avec la plate forme elle-même sans implanter d'applications d'analyse. Cela permettrait de déterminer quels types d'applications sont réalisables avec les marges offertes par notre plate-forme. D'autre part, nous avons effectué des tests de performance avec des données réelles acquises lors d'une période de 24 heures. Il faudrait aussi effectuer des tests avec des données simulées qui refléteraient une activité beaucoup plus intenses. Cela permettrait, en effet, de déterminer les limites de la capacité de notre plate-forme. De cette manière nous pourrions aussi vérifier son comportement lors des situations de crise une fois que les limites de la capacité auraient été dépassées.

BIBLIOGRAPHIE

[1] H. Yamashita et al, *A real-time data-acquisition and analysis system with distributed UNIX workstations.* Nuclear Instruments & Methods in Physics Research, Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 379 2 Sep 11 1996 Elsevier Science B.V., 276-282

[2] S.G. Belovitch, J. M. Belovitch, *A dynamically-alterable database for real-time data recording.* Proceedings of the Industrial Computing Conference Proceedings of the 1996 Industrial Computing Conference, Oct 6-11 1996, 203-216.

[3] King Todd, *Dynamic data structures: theory and application,* Academic Press Ltd 1992.

[4] Chen M-S., Han J, Yu P.S., *Data mining: an overview from a database perspective.* IEEE Transactions on Knowledge and Data Engineering Vol. 6 No. 6 December 1996, 866-883

[5] White Colin J., *Sybase adaptive server IQ.* Database Associates International 1992.

[6] Nakajima H., *Fuzzy logic and datamining,* Proceedings of the Asian Fuzzy Systems Symposium Soft Computing in Intelligent Systems and Information Processing, Proceedings of the 1996 Asian Fuzzy Systems Symposium Dec 11-14 1996, 133-138

[7] Hirota K., Pedrycz W., *Linguistic datamining and fuzzy modeling.* IEEE International Conference on Fuzzy Systems 2 Sep 8-11 1996 Sponsored by IEEE, 488-1492

[8] Pedrycz W., *Datamining and fuzzy modelling.* Biennial Conference of the North American Fuzzy Information Processing Society, NAFIPS New Frontiers in

Fuzzy Logic and Soft Computing, Jun 19-22 1996, 263-267.

[9] Baldwin J.F., *Knowledge from data using fuzzy methods.* Pattern Recognition Letters v17, n6, May 15 1996, Elsevier Science B.V. Amsterdam Netherlands, 593-600.

[10] Lin T.Y., Liu Q., Zuo X. *Models for first order logic applications to data mining.* Proceedings of the Asian Fuzzy Systems Symposium Soft Computing in Intelligent Systems and Information Processing, Proceedings of the 1996 Asian Fuzzy Systems Symposium Dec. 11-14 1996 Kenting, Taiwan, IEEE, 152-157.

[11] Chartwell Bratt, *Telecommunication telephone networks 2.* Ericsson, Televerket and Studentlitteratur 1987.

[12] Yost, Jeff et al. *Content-based visualization for intelligent problem-solving environments.* International Journal of Human Computer Studies v 46, n 4, Apr 1997, Academic Press Ltd, 409-441.

[13] *Plex C1.* Ericsson Telecom AB. 1993.

[14] Nguyen John, Vince Ragosta, *CMS 88 call path course, APT 210 1.* Ericsson Research Canada (LMC) 1997.

[15] Corbett E. *MDATA events for Package D.* Ericsson Ireland (EEI) 1997.

[16] *American National Standard EIA/TIA-136.* Telecom Industry Association 1999.

[17] *RF Guidelines for CMS 8800.* Ericsson Telecom AB 1996.

[18] *Interim Standard IS-54B.* Telecom Industry Association 1993.

[19] Bobick, Aaron F. Ivanov, Yuri A., *action recognition using probabilistic parsing.* Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition Jun 23-25 1998.

[20] A. Stolcke, *Bayesian learning of probabilistic language Models.* PhD Thesis, University of California at Berkley, 1994.

[21] P.J. Ramadge, W. M. Wonham. *Supervisory control of a class of discrete event processes.* SIAM J. Control and Optimization, 25(1), 1987, 206-230.

[22] E. De Santis, S. Di Gennaro, *A representation of discrete event dynamical systems.* Proceedings of the IEEE Conference on Decision and Control v 2 Dec 15-17, 1993, 1176-1181.

[23] J.G Thistle et al., *Feature interaction modelling, detection and resolution: A Supervisory Control Approach.* Imprimé par l'auteur.

[24] A.V. Aho, R.Sethi, J. D. Ullman, *Compilers principles, techniques and tool.* Addison Wesley 1988.

[25] Nigel P. Chapman, *LR parsing theory and practice.* Cambridge University Press 1987.

[26] Parsons Thomas W., *Introduction to compiler construction.* Computer Science Press 1992.

[27] Dick Grune, Cariel Jacobs, *Parsing techniques a practical guide.* Imprimé par les auteurs, Vrije Universitet, Amsterdam, The Netherlands, 1995.

[28] Knuth D.E, *The art of computer programming volume 1 fundamental algorithms 3rd editio.* Addison Wesley 1997.

[29] Knuth D.E., *The art of computer programming volume 2 sorting and searching 2nd edition.* Addison Wesley 1998.

[30] Sedgewick R., *Algorithms in C++.* Addison Wesley 1992.

[31] Ellis M. A. Stroustrup B., *The annotated C++ reference manual.* Addison Wesley 1994.

[32] Gamma E. et al., *Design Patterns.* Addison Wesley 1995.

[33] Levine J. R. Mason T. Brown D., *Lex & Yacc.* O'Reilly and Associates, inc.

1995.

[34] Lemone, K. A., *Design of compilers: techniques of programming language translation.* CRC Press 1992.

[35] Sethi, R., *Programming languages: concepts and constructs, second edition.* Addison Wesley 1996.

[36] Cameron Kelly Coursey, *Understanding digital PCS the TDMA standard.* Artech House Publishers 1999.

[37] Heine, Gunnar, *GSM Networks: protocols, terminology and implementation.* Artech House Publishers, 1998

[38] V. Carg, *IS-95 CDMA and Cdma 2000 cellular/PCS systems Implementation.* Prentice Hall Canada, 1999.

[39] P Minichiello, *TDMA adaptive antenna field trial in Phoenix PCS market, radio network performance results.* Ericsson Research Canada (LMC), LMC/UX-99:019 Uen, 1999.

[40] Bengtsson U., *Internet transport service - OCITS interface description, CP user.* Ericsson Sweden (ETX), 1/155 19-CNZ 221 003 Uen, REV. A, October 1994.

[41] David R. Boggs, Jefferey C. Mogul, Cristopher A.Kent, *Measured capacity of an Etherent: Myths and Reality.* Proceedings of the SIGCOMM 1988 Symposium on Communications Architectures and Protocols, ACM SIGCOMM, August 1988.

[42] Charles E. Spurgeon, *Ethernet the definitive guide.* O'Reilly and Associates, inc. 2000.

[43] Richard W. Stevens, *TCP/IP illustrated Vol. 1.* Addison Wesley, 1994.

[44] Jean Walrand, *Communication networks: a first course.* Aksen Associates 1991.

[45] H-E Erickson et al., *UML toolkit.* Wiley 1997.

[46] J. Bloomer, *Power programming with RPC.* O'Reilly and Associates, Inc., 1992.

[47] A. Askerup, *Emerging technologies for performance management.* Ericsson Research Canada (LMC), LMC/SD-97:124Uen, 1997

[48] Nadir Belbahri, *Pascal du début à la fin 3ème édition.* Inforum Inc., 1987.

[49] James Rumbaugh et al, *OMT Modélisation et conception orientées objet, édition française revue et augmentée.* Prentice Hall 1994.

[50] A. Silberschatz, P. B. Galvin, *Operating systems concepts.* Addison-Wesley 1994.

[51] J. L. Hennessy, D. A. Patterson, *Computer architecture, a quantitative approach.* McGrawHill, 1997.

[52] R. A. Elmasri, S B. Navathe, *Fundamentals of database systems second edition.* Benjamin Cummings, 1994.

[53] C. M. Aras et al, *Real-time Communication in packet-switched networks.* Proceedings of the IEEE Vol. 82, 1994, 122-138.

[54] G. F. Franklin, J. D. Powell, A.Emami-Naeini, *Feedback control of dynamic systems.* Addison-Wesley 1994.

[55] Arnold et al, *Real-time supersystems.* IEEE Transactions on computers, vol. c-31, no 5, May 1982, 385-398.