DURAND Luc
ESIEE - I5 - IF

Rapport de stage de 5eme année

05 janvier 2004 – 30 juin 2004

CEA Saclay
DSM / DAPNIA / SPP

Communication sous XDAQ pour la ferme de monitorage du calorimètre électromagnétique de l'expérience CMS

ESIEE	**CEA Saclay**
M. Thierry GRANPIERRE	M . Philippe GRAS
Cité Descartes	DAPNIA / SPP
2 boulevard Blaise Pascal	91 191 Gif sur Yvette Cedex
BP 99	
93162 NOISY LE GRAND Cedex	
Mail : t.grandpierre@esiee.fr	Mail : philippe.gras@cern.ch

TABLE DES MATIERES

INTRODUCTION

L'objectif de mon stage est la réalisation, dans l'environnement de programmation XDAQ, d'un module de communication pour la ferme de monitorage du calorimètre électromagnétique de l'expérience CMS.

L'expérience CMS forme avec ATLAS, ALICE et LHC-B les quatre principales expériences de Physique des particules qui seront menées au CERN grâce au LHC, le futur grand accélérateur de particules européen. Le précèdent grand accélérateur de particule du CERN (le LEP) ne permettant plus d'atteindre les énergies nécessaires à l'apparition de nouvelles particules (en particulier le boson de Higgs), il a été décidé de le remplacer par un nouvel accélérateur, adapté aux contraintes de ces nouvelles expériences : le LHC.

Mon travail concerne un sous détecteur du détecteur utilisé pour l'expérience CMS : le calorimètre électromagnétique. Comme pour tout appareil de mesure, les données issues de ce capteur doivent être corrigées afin de tenir compte des caractéristiques du capteur. Le flot de données des mesures atteignant les 100 TOctets / s, l'architecture matérielle et logicielle de traitement doit être adaptée.

L'objectif de mon stage est donc la réalisation d'un module de communication chargé de recevoir les données de calibrage en provenance du calorimètre pour les passer à l'application chargée de les traiter.

Je réalise ce stage sur le site du CEA de Saclay (Gif-sur-Yvette), au sein du Service de Physique des Particules. Je vais donc vous présenter le CEA avant de faire une description du LHC et de l'expérience CMS. Je vous présenterais ensuite le travail que j'ai réalisé au cours de ces 6 mois de stage.

PRESENTATION DU CEA

Le CEA (Commissariat à l'Energie Atomique) a été fondé le 18 Octobre 1945 par une ordonnance du général De Gaulle. Il s'agissait, au lendemain de la seconde guerre mondiale, de permettre à la France de gagner son indépendance dans le domaine, alors tout neuf, du nucléaire. Le rôle du CEA est alors très clair, assurer les recherches dans le domaine nucléaire pour les applications industrielles, médicales et militaires.

C'est en 1947 qu'est fondé, à Saclay, le premier centre du CEA. Il accueillera la première 'pile' nucléaire française en 1948 et le premier accélérateur de particules français en 1952. Suivront ensuite les créations d'autres centres CEA : Limeil (1954), Vaujours, Bruyère le Châtel et Marcoule (1955), Grenoble (1956), Valduc (1957, plus particulièrement destiné au recherche militaires), Cadarache (1959) et Ripault (1961).

Aujourd'hui, le CEA représente, avec ses 15 024 salariés et un budget de 2,7 milliards d'euros, un acteur majeur en matière de recherche, de développement et d'innovation. Grâce à ses relations avec les industriels, ses ressources humaines (ingénieurs, chercheurs…) et matérielles (lasers, supercalculateurs, réacteurs de recherche…) est l'un des plus importants organismes européens de recherche technologique.

Comme présenté sur cet Organigramme du CEA, le CEA s'organise autour de quatre pôles :

- Pôle défense
- Pôle recherche
- Pôle recherche technologique
- Pôle nucléaire

Le centre de Saclay fait partie du pôle nucléaire.

Centre de Saclay

Le centre de Saclay reste le centre historique du CEA et un acteur important de la recherche nucléaire avec ses deux réacteurs expérimentaux : Osiris et Orphée (voir l'Organigramme du CEA en annexe A).

Sur le site de Saclay, travaillent environ 6 000 personnes, employés du CEA, collaborateurs détachés d'autres laboratoires ou d'entreprises extérieures ou étudiants. De nombreuses spécialités sont représentées : Sciences du vivant, Physiques des particules, Physiques des états condensés, Astrophysique, chacune d'entre elles dépendant d'une direction.

Durant mon stage, j'ai travaillé au sein du service de physique des particules dans l'équipe CMS / ECAL.

DSM

Comme montré sur cet Organigramme de la DSM, en annexe B, la Direction des Sciences de la Matière est en charge de toutes les recherches ayant pour thème l'exploration de la matière. Elle comporte plusieurs départements :

- DAPNIA : Département d'Astrophysique, de physique des particules, de physique Nucléaire et de l'Instrumentation Associée
- DRECAM : Département de Recherche sur l'Etat Condensé, les Atomes et les Molécules
- DRFMC : Département de Recherche Fondamentale sur la Matière Condensée
- LSCE : Laboratoire des Sciences du Climat et de l'Environnement
- SPhT : Service de Physique Théorique
- GANIL : Grand Accélérateur National d'Ions Lourds

DAPNIA

Le Département d'Astrophysique, de physique des Particules, de physique Nucléaire et de l'Instrumentation Associée regroupe lui aussi plusieurs services. Trois services de Physique :

- SAp : Service d'Astrophysique
- SPP : Service de Physique des Particules
- SPhN : Service de Physique Nucléaire

Et quatre services techniques :

- SDA : Service de Déclassement des Accélérateurs
- SACM : Service des Accélérateurs, de Cryogénie et de Magnétisme
- SEDI : Service d'Électronique, des Détecteurs et d'Informatique
- SIS : Le Service d'Ingénierie des Systèmes

Le Service de Physique des Particules participe à de nombreuses expériences, couvrant tout le domaine de la physique des particules élémentaires. Il est impliqué dans plusieurs des expériences du LHC au CERN, et en particulier les expériences CMS et ATLAS.

PRESENTATION DU SUJET DU STAGE

Le but de ce stage est le développement et l'intégration d'un module de communication pour la ferme de monitorage du calorimètre électromagnétique de l'expérience CMS. Je vais donc commencer par la présentation du LHC, puis de l'expérience CMS dans sa globalité avant de terminer par la présentation précise du sujet de stage et de son contexte.

Le LHC

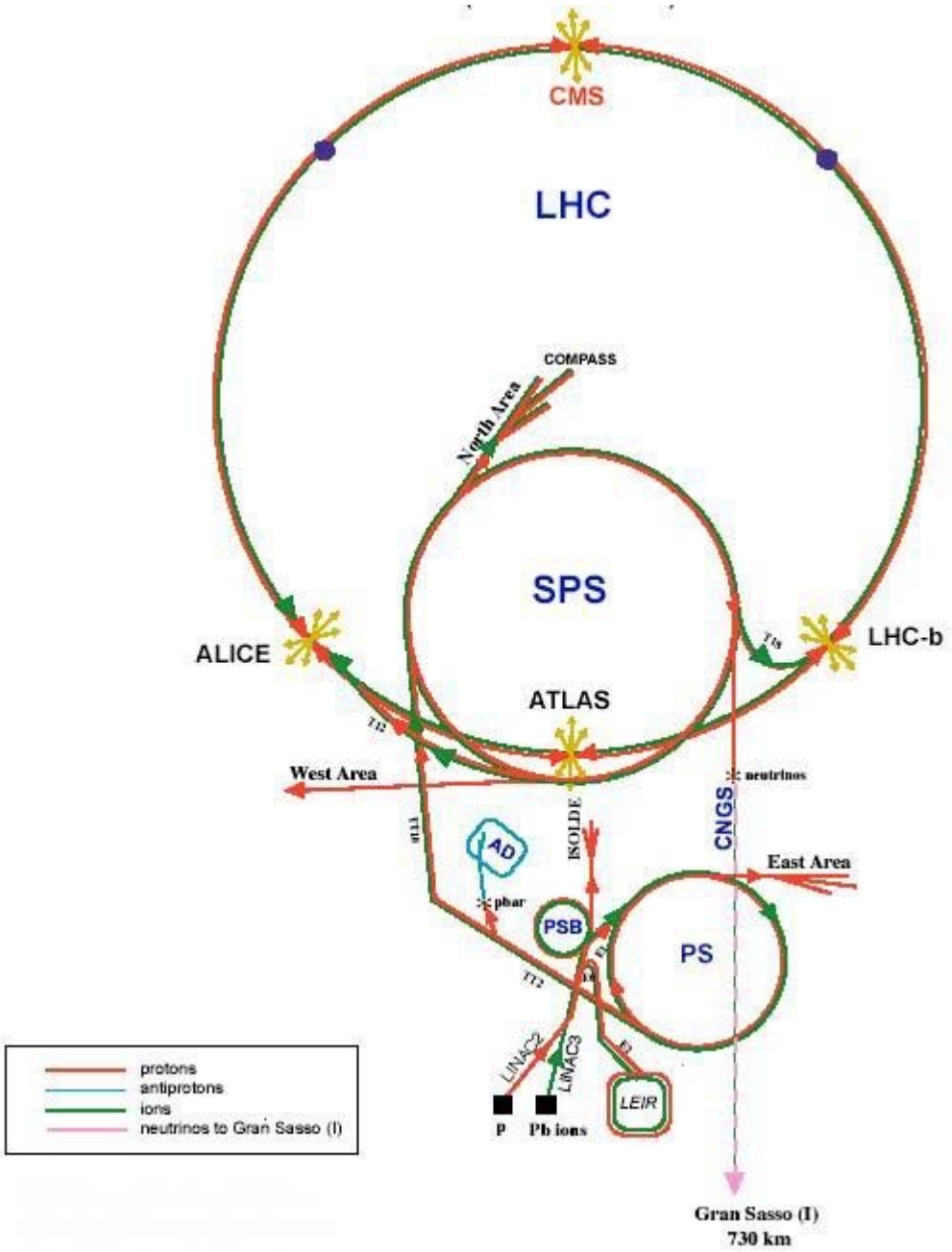
Le LHC (Large Hadron Collider : grand collisionneur de hadrons[1]) est, comme son nom l'indique, un grand accélérateur de particules. Il sera capable d'accélérer, et de faire se collisionner plusieurs types de particules :

- protons, avec une énergie de 14 TeV dans le centre de masse
- ions lourds (des noyaux d'atome de plomb), avec une énergie de 1148 TeV dans le centre de masse

Afin d'atteindre ces énergie très élevées, plusieurs niveaux d'accélérateurs seront successivement utilisés :

- deux accélérateurs linéaires (LINAC : LINear ACcelerator) seront chargés de l'injection des protons et des ions lourds dans le PS
- un synchrotron (PS : Proton Synchrotron) sera chargé du stockage des protons en provenance des LINAC avant leur injection dans le PSB
- un synchrotron booster (PSB : Proton Booster Synchrotron) sera chargé d'accélérer encore les protons provenant du PSB avant leur injection dans le SPS
- un super synchrotron (SPS : Super Proton Synchrotron) sera chargé d'accélérer une nouvelle fois les protons afin de les amener à une vitesse proche de celle de la lumière avant leur injection dans le LHC.
- le LHC (l'anneau final de 27 Km de circonférence), recevra les protons pour les stocker et les faire se collisionner aux endroits des expériences

1 Un hadron est une particule composée de quarks et sensible à l'interaction forte (voir le Lexique de la physique des particules, en annexe G). Dans le cas présent, il s'agit de protons.



Dans le LHC, un anneau de 27 Km de circonférence, les particules circulent simultanément dans les deux sens. A chacun des deux sens de rotation correspond un tube à vide dans lequel se déplacent les paquets de particules. Les particules étant guidées par d'intenses champs magnétiques, générés par des aimants supraconducteurs, l'ensemble de l'appareillage est maintenu à une température de 2 K (– 271° C).

Comme les particules se déplacent par paquets, lorsque l'on fait se croiser les deux faisceaux, on obtient alors des rafales de collisions, séparées les unes des autres par 25 ns. On parle alors de bunch crossing.

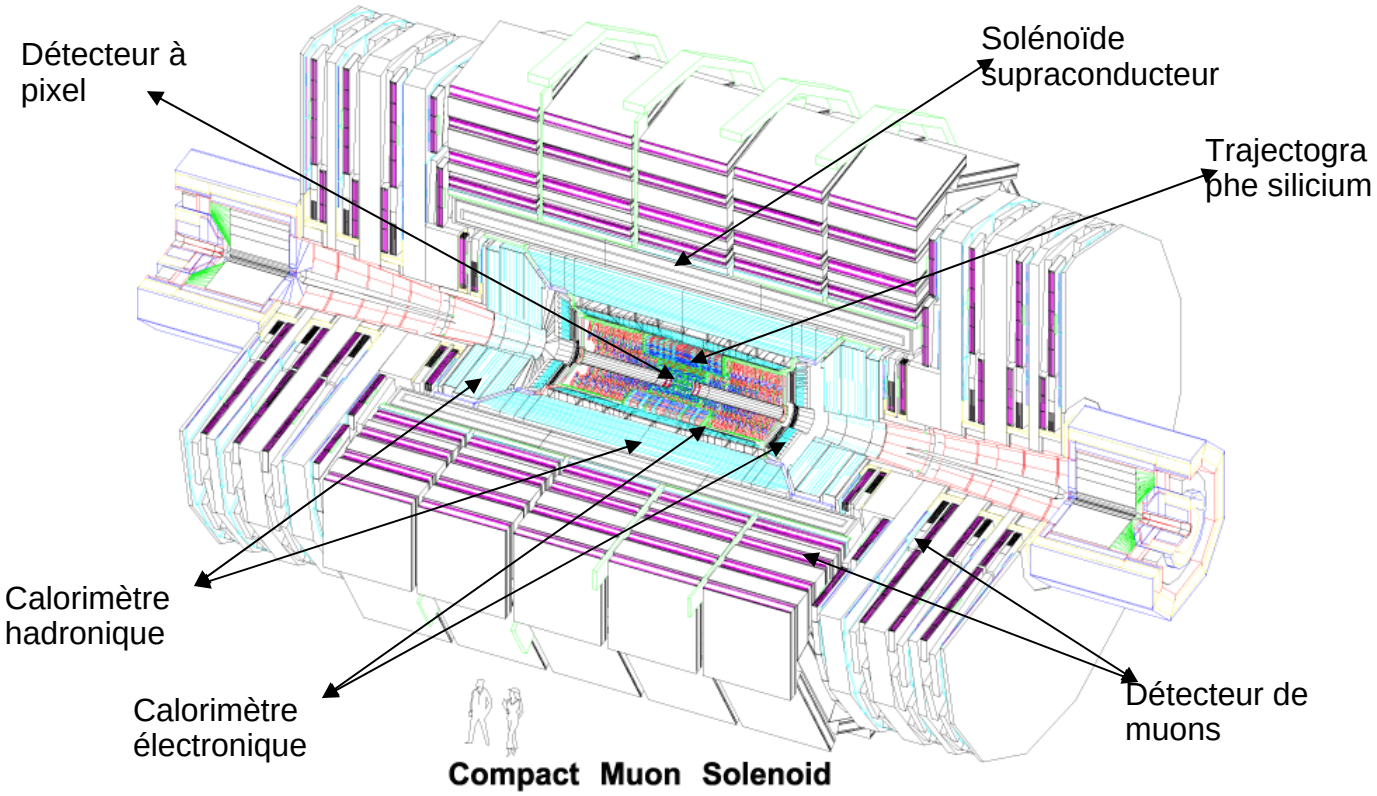
Le détecteur CMS

A chacun des quatre endroits où se produisent les collisions est installée une expérience et un détecteur :

- CMS et ATLAS: ces détecteurs sont principalement destinés à la mise en évidence du boson du Higgs et à l'étude de la Physique au-delà du modèle standard
- Alice : ce détecteur est destinés à l'étude des plasmas quarks / gluons, dans les collisions d'ions lourds
- LHC-b : ce détecteur est destiné à l'étude de la violation de la symétrie CP.

Constitution

Le détecteur CMS est composé de plusieurs sous-détecteurs et d'un aimant (solénoïde).

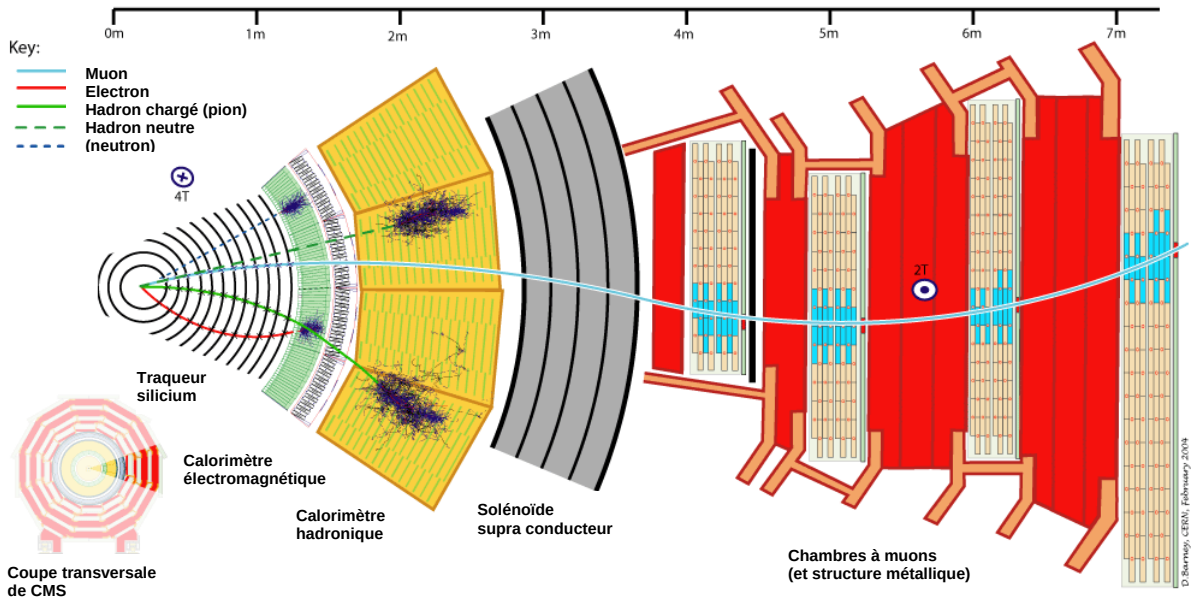


Chacun des détecteurs possèdent une fonction précise :

- Détecteurs de muons : ils sont chargés de suivre la trajectoire des muons
- Calorimètre hadronique : il mesure la position et l'énergie des hadrons (en les détruisant)
- Calorimètre Electronique : il mesure la position et l'énergie des photons et des électrons (en les détruisant)
- Trajectographe silicium : il mesure la trajectoire des particules chargées (sans les affecter)

Le rôle du solénoïde supraconducteur est de produire un champs magnétique intense (4T au cœur du détecteur, 2 T sur la périphérie) de façon à courber la trajectoire des particules chargées et ainsi permettre leur

identification (charge et masse). Voici des exemples de trajectoires de certaines particules dans le détecteur :

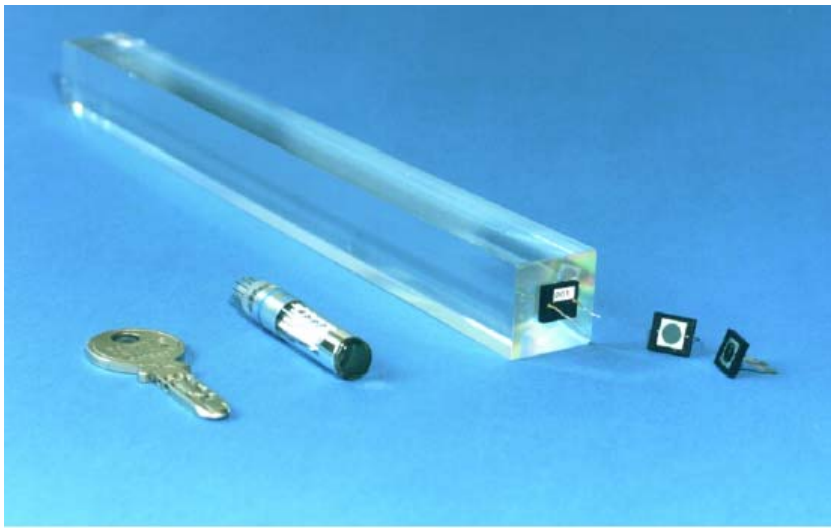


Les neutrinos, interagissant très faiblement avec la matière, ne peuvent être détectés. Leur caractérisation est obtenue à partir du bilan d'énergie global de la collision.

Calorimètre électromagnétique

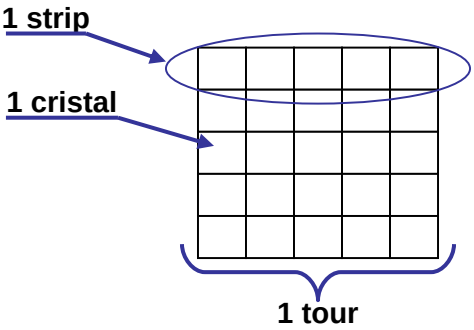
Le calorimètre électromagnétique, appelé ECAL (Electronic CALorimeter), permet de mesurer l'énergie des électrons et des photons émis lors d'une collision.

La mesure de cette énergie exploite l'effet de photoluminescence produite lors de l'interaction d'une particule avec un cristal. L'interaction de l'électron avec les atomes du cristal provoque l'émission de photons. Le calorimètre est composé de 80 00 cristaux de Tungstate de Plomb. Chacun des cristaux est pourvu à une extrémité d'une photodiode à avalanche (APD : Avalanche PhotoDiode) ou d'une phototriode à vide qui permet de mesurer la quantité de lumière émise dans le cristal. De cette mesure est déduite l'énergie déposée par l'électron.



En récoltant toutes les mesures fournies par ces photodiodes, on peut donc obtenir les énergies de chacun des électrons émis par la collision et leurs positions au moment de leurs interactions avec le calorimètre. En corrélant ces mesures avec celle du trajectographe, on peut donc obtenir la trajectoire et l'énergie de chacun de ces électrons.

La lecture des 80 000 cristaux qui composeront ECAL est structurée de la manière suivante. Les cristaux sont regroupés en strip (5 cristaux alignés), eux même regroupés en tours (5 strips consécutifs). La tour constitue l'unité de base de la lecture du calorimètre : lors d'une lecture, on obtient au minimum les 25 canaux correspondant aux 25 cristaux d'une tour. Les tours sont ensuite organisées en modules puis en super-modules (68 tours, soit 1700 cristaux).



Lors de la lecture des cristaux, plusieurs optimisations sont possible pour réduire la quantité de données transmises et économiser de la bande passante :

- Suppression des canaux ne comportant pas d'information (Zéro Suppression).
- Lecture sélective d'une partie des tours seulement (Selective Readout).

Calibrage ECAL

Le calorimètre électromagnétique est soumis à un niveau de radiations élevé. Ces radiations finissent par modifier la structure même des cristaux et affectent leurs propriétés optiques (indice de réfraction, transparence…) et donc les mesures. D'autres facteurs tels que les disparités entre composants, la température, les courants de fuites des diodes… peuvent aussi les affecter.

Pour conserver la meilleure précision possible dans les mesures, il faut donc régulièrement calibrer le calorimètre. Pour ce faire, plusieurs types de mesures sont prévus sur les cristaux, les photodiodes et leur environnement :

- Mesure de température : des sondes de températures permettent de suivre la température des cristaux et des dispositifs électroniques de lecture.
- Mesure de courant de fuite : les diodes utilisées n'étant pas idéales, une mesure du noir (en l'absence de rayonnement) est nécessaire.
- Mesure de transparence des cristaux : les radiations altèrent la transparence des cristaux. Un faisceau laser (trois longueures d'ondes sont disponibles) est envoyé sur les cristaux et sur une photodiode. En comparant l'intensité lumineuse reçue par la photodiode du cristal (celle servant à la mesure de l'énergie déposée par les électrons) avec celle reçue par la photodiode témoin (qui mesure l'intensité du faisceau laser), on peut mesurer la transparence des cristaux et le gain du dispositif.

Afin de permettre un calibrage le meilleur possible, deux phases de calibrage seront mises en place :

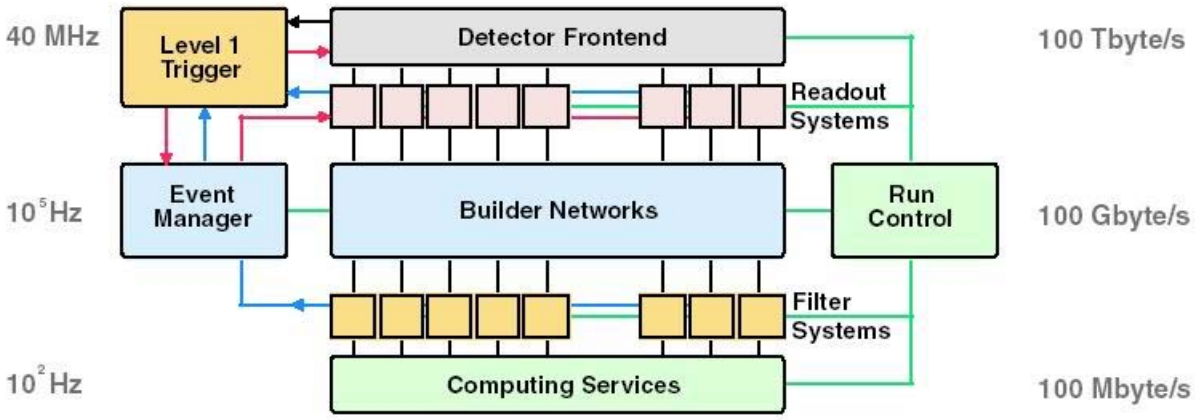
- Calibrage précis : au démarrage du détecteur et ensuite de façon régulière, des séquences uniquement réservées au calibrage seront programmées. Durant ces séquences, tous les cristaux seront testés avec les trois longueurs d'ondes laser. Ces mesures serviront de base pour les calibrages suivants.
- Calibration en marche : le calibrage sera fait en continu, durant l'acquisition des données de Physique. Les mesures se font par séquences, sur un demi super module à chaque fois. L'ensemble du calorimètre étant calibré en environs 30 minutes. Afin de pas perturber les mesures physique, les mesures se font pendant les pause d'abandon du LHC qui durent 3 µs et se reproduisent toutes les 90 µs.

Une ferme de PCs, appelée « ferme de monitorage », calculera en ligne à partir de ces mesures les corrections qui devront être appliquées aux mesures physiques. Ces corrections permettront de tenir compte du vieillissement des cristaux et des variations des conditions de mesures.

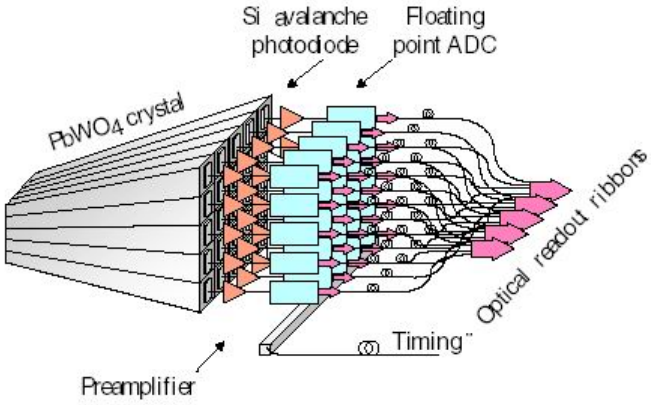
Acquisition des données

L'acquisition et le traitement des données (en phase d'acquisition ou en phase de calibrage), est un autre grand défit. Les collisions se produisant toutes les 25 ns, le dispositif électronique de mesure doit être capable de traiter les 80 000 canaux de données (un par cristal) 40 millions de fois par seconde, pour ne conserver que les événements intéressants (environs 1 par seconde).

L'énorme flot de données ainsi généré a nécessité la mise en place d'une structure de communication adaptée. Chaque étage de traitement devra réduire les flots de données obtenues avec des traitements des plus en plus complexes pour arriver à ne conserver que les événements potentiellement intéressants. Ces événements seront stockés sur support informatique avant d'être traité à grande échelle par des grilles de calcul.



Trigger and Data Acquisition baseline structure



Les dispositif de numérisation des données seront situés au plus prés des cristaux (VFE : Very Front End). Ils se chargeront de la transformation des mesures des photodiodes en données numériques qui seront envoyées par fibres optiques au niveau de traitement suivant.

Etant donné les doses de radiations subies par ces équipements, ceux-ci doivent être durcis aux radiation (RadHard : Radiation Hard) et sont donc beaucoup plus onéreux. Pour minimiser les coûts et les problèmes liés aux radiations, l'étage de traitement suivant est déporté et se trouve dans une salle séparée de celle ou se trouve le capteur.

L'étage de traitement suivant est chargé de sélectionner parmi tous les événements captés ceux qui présentent un intérêt, de façon à diminuer la quantité de donnée générée d'un facteur 1000. Cette sélection, qui doit se faire très rapidement, est basée sur des critères simples et sur des données ne provenant que du seul ECAL. C'est le rôle du Trigger de niveau 1. Il permet, de plus, de lire de façon précise une région limitée du calorimètre (pour améliorer la précision de lecture) tout en conservant une vision globale de l'énergie déposée.

L'étage suivant réalise une nouvelle sélection, mais en utilisant, cette fois ci, les données provenant de tout les capteurs de CMS. Cette sélection est elle aussi faite en hardware sur des carte spécialisée à base de FPGA (DCC : Data Control Card). Les données obtenues sont ensuite stockées dans des mémoires tampon avant leur récupération par le niveau de traitement suivant.

La dernière sélection est ensuite faite par une ferme d'ordinateurs qui disposeront de plusieurs dizaines de millisecondes pour leurs traitements et qui pourront donc employer des algorithmes plus évolués. Les événements restants seront ensuite envoyés, via un réseau gigabits, au service de stockage du CERN. Le traitement de ces données sera ensuite fait, en

différé par une grille de calcul, afin de répondre aux énormes besoins en puissance de calcul des traitements nécessaires. Ce dispositif constitue le système d'acquisition (DAQ System : Data AcQuisition).

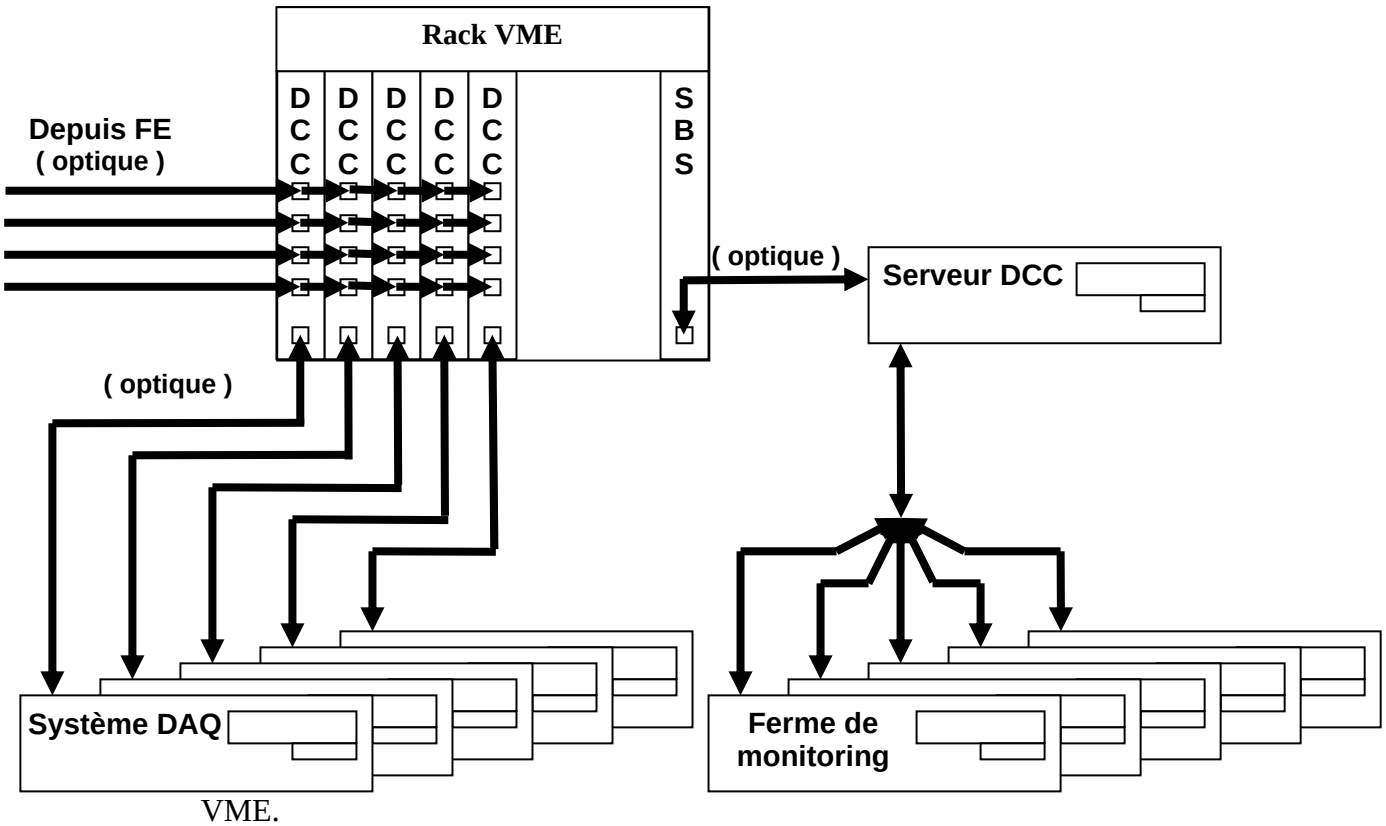
En parallèle de cette ferme de traitement, une autre ferme d'ordinateurs sera utilisée pour traiter les données de calibrage (monitoring farm : ferme de monitorage). Lors des séquences de calibrage, les données obtenues par l'acquisition seront récupérées par la ferme de monitoring. Ces données seront traitées en temps réel pour obtenir les valeurs de corrections qui seront appliquées aux mesures.

Objectifs du stage

Le sujet de mon stage concerne le détecteur CMS, et plus précisément les systèmes d'acquisition et de traitement des données de monitorage du calorimètre ECAL.

Localisation du projet

Les données produites par l'électronique Front End sont récupérées, traitées et stockées par les cartes DCC. Les données FE arrivent directement sur les cartes DCC via des liens optiques. Toutes les cartes DCC, d'un même rack, sont reliées entre elles par un bus de fond de panier de type VME.



Les données à destination du système DAQ sont récupérées par des liens optiques, directement depuis chacune des cartes DCC. Le système DAQ étant chargé du stockage de toutes les données en provenance de CMS, il recevra les données d'acquisition.

Les données de monitorage utilisent une autre voie. Elles sont tout d'abord collectées par une carte de communication, via le bus VME, et sont ensuite envoyées sur un PC serveur par un lien optique de type SBS. Les PCs de la ferme de monitoring récupèrent ensuite les données depuis ce serveur DCC par un réseau Ethernet.

Il s'agit là du chemin qu'utiliseront les données dans la configuration final de CMS. En attendant, pour les mesures qui seront effectuées durant les faisceaux de test, une autre configuration sera utilisée. Comme le nombre de données à traiter sera très inférieur à celui de la configuration finale, seul seront utilisés, un PC pour le serveur de données DCC, un PC pour le système DAQ (en fait un PC fixe et plusieurs PCs embraqués) et un PC pour le système de monitoring.

Les données reçues par les cartes DCC depuis l'électronique FE seront toutes (acquisition et monitorage) envoyées sur le serveur DCC via le bus VME et le lien optique SBS.

Le serveur DCC se chargera de la séparation des données d'acquisition et des données de monitorage. Les données de monitorage seront envoyées sur le PC de monitorage et les données d'acquisition et de monitorage seront envoyées sur le PC DAQ par un réseau Ethernet.

Objectifs du stage

Le sujet de ce stage concerne la communication entre le serveur DCC et la ferme de monitoring. Ses objectifs sont :

- Tester la communication entre le serveur DCC et les PCs de monitoring et du système DAQ.
  Il s'agit de s'assurer que les liens de communication seront suffisants pour le débit de données attendu lors du faisceau de test et lors du fonctionnement final.
- Développer le logiciel qui réalisera la communication entre le serveur DCC et la ferme de monitoring.
- Modifier le logiciel de monitorage de façon à ce qu'il s'interface avec le module de communication et les classes de décodage du format de données.
- Tester et mesurer les performances du système complet (serveur DCC, module de réception, classes de décodage, application d'analyses).
- Il s'agit de vérifier que les performances du système seront suffisantes pour être supportées par la configuration de fonctionnement.

Environnement matériel et logiciel

Afin de simuler, à échelle réduite, ce qui sera la ferme de monitoring et le système DAQ, je dispose :

- De deux serveurs rackables, du même type que les PCs qui seront effectivement utilisés dans la ferme de monitoring, de type PC x86 (bi Pentium 4 Xeon / 1GO de RAM) : dapvro1 et dapvro2
- D'une machine de bureau, de PC x86 (Pentium 4 / 512 MO de RAM) : daplxa176

Ces trois machines sont placées dans une salle d'expérimentation et sont donc utilisée de façon distante, par l'intermédiaire du réseau global du DAPNIA. Toutes ces machines fonctionnent sous Linux Red Hat 9. Tous mes développements ont donc été fait avec les systèmes fournit en standard sur les distributions Linux Red Hat :

- Compilateur C / C++ : gcc 3.3.2
- Gestionnaire de Makefile : Gnu Make
- Compilateur de documentation : Doxygen
- Framework graphique : Qt 3.2.2

Dans un souci d'uniformisation et de simplification des communications dans le projet CMS, un framework de gestion de processus et de gestion de communication interprocessus est utilisé. Ce framework : XDAQ (Trigger and Data AcQuisition), est développé au CERN est doit être utilisé par tout les composants logiciels qui interviendront dans le traitement des données sur CMS.

TRAVAIL REALISE

PRISE EN MAIN DE XDAQ

Découverte et premier module

Pour découvrir les bases de XDAQ et son mode d'utilisation, j'avais à ma disposition des exemples fournis avec leurs sources :

- HelloWorld : la version XDAQ du célèbre logiciel qui n'a pour autre fonction que d'afficher « Hello world » à l'écran.
- Benchmark : une suite d'exemples d'utilisation des modes de communications XDAQ.

Et les guides (appelés compagnons) d'utilisations :

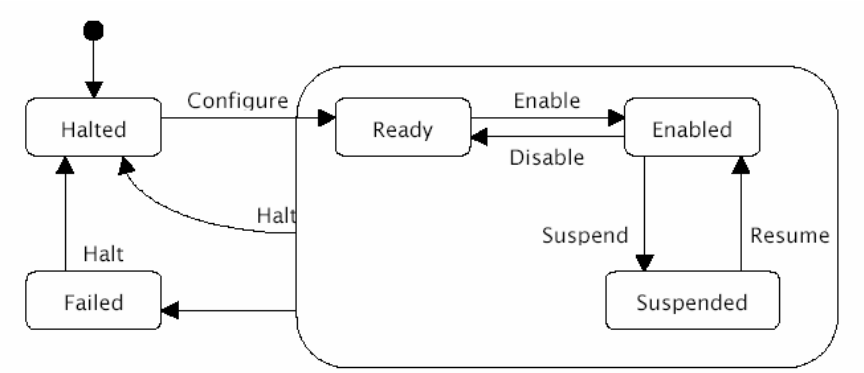
- Getting started : installation et première utilisation de XDAQ via l'exemple du HelloWorld
- Writting applications : description des l'utilisation du Logger (enregistrement et affichage de message d'informations ou d'erreurs), principe de la numérotation des versions, utilisation et transition entre les divers états d'un processus.
- Configuration : utilisations des fichiers de configurations et de l'interface graphique XDAQWin
- I2O messaging : description et principes d'utilisation du protocole de communication I2O
- SOAP messaging : description et principes d'utilisation du protocole de communication SOAP

Processus XDAQ

XDAQ se présente sous la forme d'un exécutable (xdaq.exe) accompagné de librairies. Les programmes de l'utilisateur se présentent sous la forme de modules venant se charger dans l'exécutable XDAQ. Un module utilisateur apparaît donc comme un « processus » contrôlé par l'exécutable et peut prendre plusieurs états :

- Halted (arrêté) : le processus est stoppé, il ne reçoit ou n'émet aucune communication et ne consomme aucun temps processeur
- Ready (prêt) : le processus est initialisé mais ne peut ni recevoir ni émettre de communication
- Enabled (actif) : le processus est pleinement fonctionnel, il peut émettre et recevoir des communications
- Suspended (en pause) : le processus peut recevoir des communications, mais il peut en émettre
- Failed (en erreur) : le processus a été stoppé car une erreur s'est produite.

Un certain nombre de messages permettent de faire changer l'état du processus :



Tous les paramètres relatifs à l'état du processus sont exportés par l'exécutif XDAQ. Un autre processus XDAQ, éventuellement chargé dans un autre exécutif XDAQ, sur une autre machine, peut ainsi savoir l'état de tous les processus XDAQ présents sur son réseau. Les processus XDAQ peuvent même se contrôler les uns les autres.

Communications XDAQ

Les communications sous XDAQ utilisent deux protocoles de haut niveau :

- SOAP (Simple Object Acces Protocole) : il s'agit d'un protocole de communication objet échangeant des messages au format XML.

De part son format très structuré, ce format est plutôt destiné à l'échange de messages courts comme des commandes. Un message SOAP sera de la forme :

```
<SOAP-ENV : Envelope>
        <SOAP-ENV : Header>
        </SOAP-ENV : Header>
        <SOAP-ENV : Body>
                <Command originator='' '' targetID='' ''/>
                <Fonction>
                        Valeur
                </Fonction>
        </SOAP-ENV : Body>
</SOAP-ENV : Envelope>
```

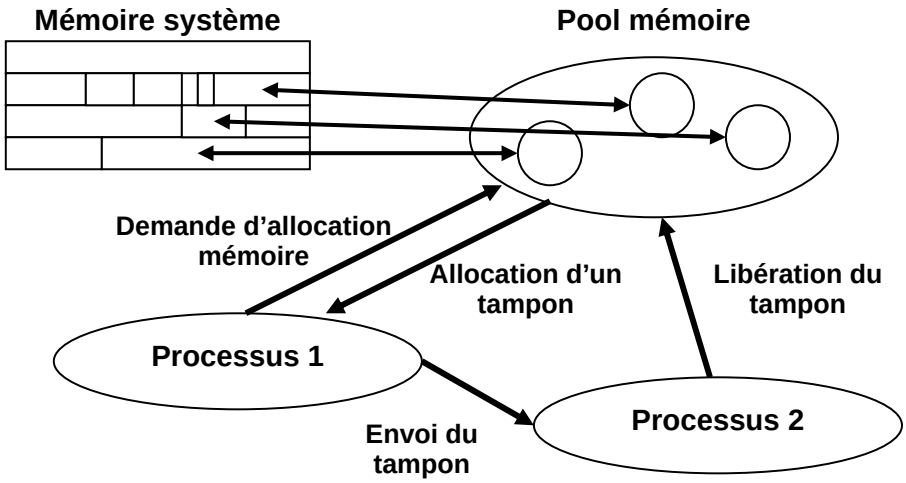
Ces informations étant encapsulées dans un message au format MIME, il est possible d'ajouter au message un nombre quelconque de fichier attachés d'un type quelconque.

Ce protocole est essentiellement utilisé pour envoyer des commandes et configurer les processus XDAQ.

- I2O (Intelligent Input Output) : il s'agit d'un protocole de communication de blocs de données (avec un formatage utilisateur).

Il a été conçu à l'origine pour permettre des communications entre matériel et système d'exploitation, tout en restant indépendant du système utilisé. Il est plutôt destiné au passage de blocs de données brutes (sans formatage), dans les cas ou les performances de communications sont critiques.

La gestion mémoire de XDAQ, pour les communications, est basée sur le principe du pool de mémoire.



Tous se passe comme si tous les exécutifs XDAQ disposaient d'un pool de mémoire partagé entre eux. Ce pool de mémoire s'adapte automatiquement en taille pour répondre au besoin des applications.

Lors d'une communication, le processus fait une allocation d'un tampon de mémoire qu'il remplit avec des données. Le tampon de mémoire est ensuite passé au processus cible de la communication qui en extrait les données. Le tampon est enfin libéré par le processus cible et retourne dans le pool.

Les processus de communications sont masqués, du point de vue des modules XDAQ tout se passe comme s'il s'agissait de passage de pointeurs sur des tampons de mémoire. Ce principe permet d'éviter la fragmentation de la mémoire et limite le nombre des appels d'allocations ou de libération mémoire. En contrepartie, une limite maximale est imposée sur la taille des tampons de mémoires : 256 KOctets. S'il est nécessaire de transporter une taille plus importante de données, celles-ci devront être reparties sur plusieurs tampons mémoires.

La répartition des données sur plusieurs tampons mémoire reste à la charge du programmeur. Il est possible de chaîner les tampons mémoires de façon à conserver l'ordre de données et d'envoyer les données d'un processus à un autre en une seule commande. Néanmoins, il convient de faire attention au fait que des tampons envoyés chaînés n'arriveront chaînés que si la communication est locale (sur la même machine). Dans le cas contraire les tampons de la chaîne arriveront dans l'ordre mais non chaînés et le programmeur aura la charge de reconstruire les données.

Plusieurs protocoles de communication sont utilisables pour le transfert effectif des données :

- FIFO : ce protocole de communication utilise un tampon FIFO pour le passage des données, il n'est donc utilisable qu'en local
- TCP : il s'agit du protocole TCP / IP
- ATCP : il s'agit d'un implémentation asynchrone du protocole TCP / IP

- MAZE : il s'agit d'un protocole de communication spécialisé pour les réseaux Myrinet.

XDAQWin

Le logiciel XDAQWin permet de configurer et de contrôler un groupe d'exécutifs XDAQ (et les modules chargés) de façon distante. Il se présente sous la forme d'une interface graphique java dans laquelle l'utilisateur doit charger un fichier de configuration.

Ce fichier de configuration, au format XML permet de décrire l'organisation des machines du réseau, leur configuration et la configuration des modules XDAQ à charger.

```
< ?xml version='1.0' ?>
<Partition>
	<Definitions>
		<ClassDef id='…'>
			…
		</ClassDef>
	</Definitions>
	<Host id='…' url='http://… :…'>
		<Address type='…' hostname='…' port='…' network='…'/>
		<Application class='…' targetAddr='…' instance='…'>
			<DefaultParameters name='Parameters' type='bag'>
				<Parameters name='…' type='…'>
					…
				</Parameters>
				…
			</DefaultParameters>
			…
		</Application>
		<Transport class='…' targetAddr='…' instance='…'>
		<urlApplication>
			…
		</urlApplication>
		<urlTransport>
			…
		</urlTransport>
	</Host>
	…
</Partition>
```

Une partition (<Partition/>) représente un groupe d'ordinateurs reliés par un réseau de communication.

Les définitions (<Definitions/>) permettent de spécifier les noms des modules XDAQ qui seront utilisés sur cette partition. Pour chaque nom de module, on définit ainsi un identifiant unique.

Chaque section de définition (<Host/>) d'un hôte permet de définir un exécutif XDAQ (plusieurs exécutifs XDAQ pouvant tourner simultanément sur une même machine).

L'adresse (<Address/>) permet de spécifier le nom de l'hôte, le type de protocole de communication à utiliser et le nom du réseau virtuel qui sera utilisé. Il est en effet possible de créer, au dessus du réseau physique, un système de réseaux virtuels permettant de regrouper les hôtes de façon logique. Le protocole spécifié (ptTCP, ptFIFO…) sera utilisé pour la communication entre les exécutifs XDAQ.

L'application (<Application/>) permet de spécifier la configuration du module qui sera chargé dans l'hôte. Le choix du module se fait en spécifiant son nom, tel qu'il est défini dans la section de définition des modules. Chaque module chargé se verra ainsi attribuer un numéro d'instance unique et une adresse (utilisée pour les communications entre modules XDAQ).

Il est possible de spécifier les valeurs par défauts des différents paramètres exportés par un module (<DefaultParameters/>). XDAQWin sera alors en mesure de les initialiser au chargement du module et éventuellement de les réinitialiser en cours de fonctionnement. Il faut, pour chaque paramètre exporté, définir un nom, un type et une valeur par défaut.

Le protocole de communication (<Transport/>) permet de configurer le module de communication qui sera utilisé par l'exécutif XDAQ. De même que pour le module utilisateur, il faut définir le nom du module à utiliser (ptTCP…), une adresse et un numéro d'instance unique.

Pour chacune des classes définies dans la section de définition des classes, il faut ensuite spécifier le chemin local permettant d'accéder au module. Pour le module utilisateur, il s'agit de la section url de l'application (<urlApplication/>), pour le module de transport, il s'agit de la section url du protocole de communication (<urlTransport/>).

Une fois ce fichier de configuration chargé, XDAQWin permet de configurer à distance tous les exécutifs XDAQ de la partition. Il permet aussi de contrôler l'état des modules (chargement, mise en pause…) et de consulter ou de modifier les paramètres exportés par les modules.

Difficultés rencontrées

XDAQ reste un logiciel en cours de développement. Il existe donc un décalage assez important entre la documentation disponible et les fonctionnalités exactes de la dernière version disponible.

La première difficulté que j'ai rencontrée, après installation, fut de faire tourner l'exemple HelloWorld. Le code source proposé été correct mais le fichier de configuration ne fonctionnait plus avec ma version de XDAQWin. J'ai donc dû corriger les manques du fichier de configuration donné en exemple (taille des vecteur dans les paramètres par défaut) afin de pouvoir charger l'exemple.

Tout au long des mes développement et de mes test, je me suis heurté au fait que la documentation fournie est très parcellaire. De nombreuses fonctions sont évoquées brièvement mais aucune explication précise ne permet de les utiliser.

Par exemple, lorsque que j'ai voulu utiliser des quantité de données supérieur à la limite des tampons mémoires (256 KOctets), la documentation ne faisait qu'évoquer la possibilité d'un chaînage des tampons ou l'utilisation d'une fonctionnalité SGL du protocole I2O. Après un échange de mail avec le responsable du développement de XDAQ, il s'averre que la fonctionnalité SGL avait été abandonnée et que seule restait le chaînage de tampons. J'ai du compulser les fichiers sources afin de trouver le fonctionnement exact du chaînage des tampons.

MESURE DES PERFORMANCES DE XDAQ

XDAQ n'ayant pas encore été utilisé dans le cadre du calorimètre, aucune mesure de performances n'était disponible pour s'assurer de la bonne adéquation entre XDAQ et les besoins. La première chose à faire

avant de pouvoir commencer à utiliser XDAQ était de s'assurer que ses performances seraient suffisantes pour couvrir les besoins. J'ai donc réalisé des séries de tests destinés à mesurer ces performances, dans des conditions les plus proche possibles de celles de sont utilisation normales.

Dans le cadre de CMS, le protocole I2O sera utilisé pour toutes les communications de données et le protocole SOAP pour les passages de commandes. Les communications les plus critiques, en termes de performances, seront les échanges de données au format I2O. Mes tests sont donc principalement orientés sur l'évaluation des performances des communications via le protocole I2O.

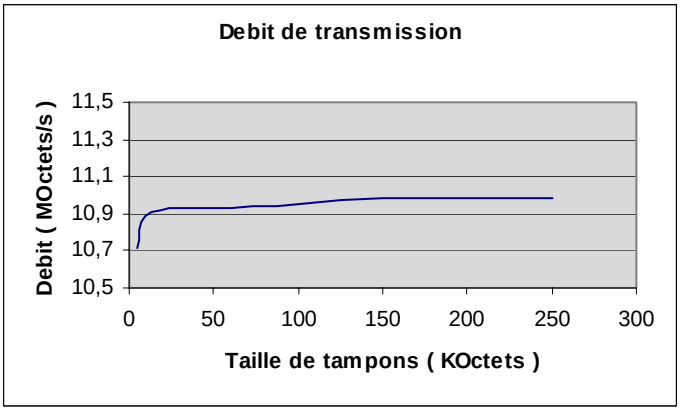
Pour ces tests, j'ai développé deux modules XDAQ :

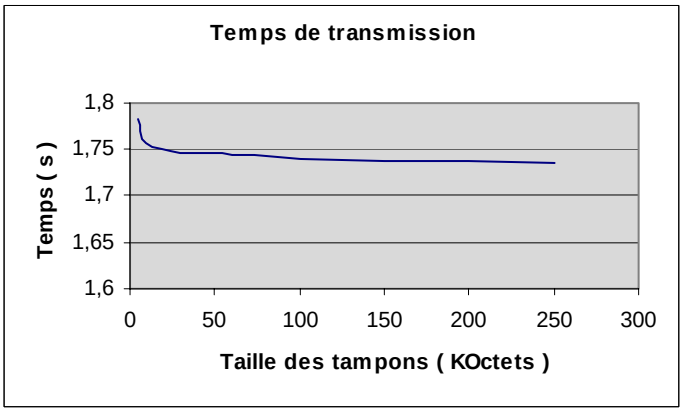
- Talk2U : le module de communication proprement dit. Il sera chargé de l'émission et la réception des données de test par le protocole I2O ainsi que des diverses mesures désirées (temps de transmission, bande passante…). Il est télécommandable par des messages SOAP. Une instance de ce module sera chargée sur chacune des machines du réseau à tester.
- Talk4U : le module de commande des modules Talk2U. Il permet de configurer les tests par l'intermédiaire d'une interface graphique Qt ou du logiciel XDAQWin. Il commande les modules Talk2U et récupère les résultats des mesures par le protocole SOAP.

Mesures XDAQ

Pour les tests de communication en simple sens (avec des données ne circulant que dans un sens à la fois sur la liaison), le processus émetteur construit une chaîne de tampons mémoire XDAQ dont la taille total des données est spécifiée par l'utilisateur. Cette chaîne est ensuite envoyée au processus récepteur. Le processus récepteur attend d'avoir reçus tous les tampons des données avant de les renvoyer à l'expéditeur.

Le processus émetteur mesure le temps total séparant le début de l'émission du premier tampon de la fin de la réception du dernier tampon. Il calcul ensuite le débit de transmission, l'overhead[2].





[2] Lors d'une transmission par réseau, en plus des données utilisateur, un certain nombre de données sont présentes pour permettre le bon fonctionnement du protocole de communication. L'overhead d'une transmission représente le rapport entre les données utilisateurs et les données supplémentaires transmises. Cela représente les frais de transmission, en terme de quantité de données.

On constate que la saturation de la connexion intervient pour des paquets de taille supérieure à 20 KOctets. En dessous de cette taille, l'overhead devient plus important et les temps de transmission chutent. La taille des données d'entête nécessaire au protocole de transmission reste toujours la même, un diminution de la quantité de données envoyées par paquets se traduit par une augmentation de la proportion de données d'entête.

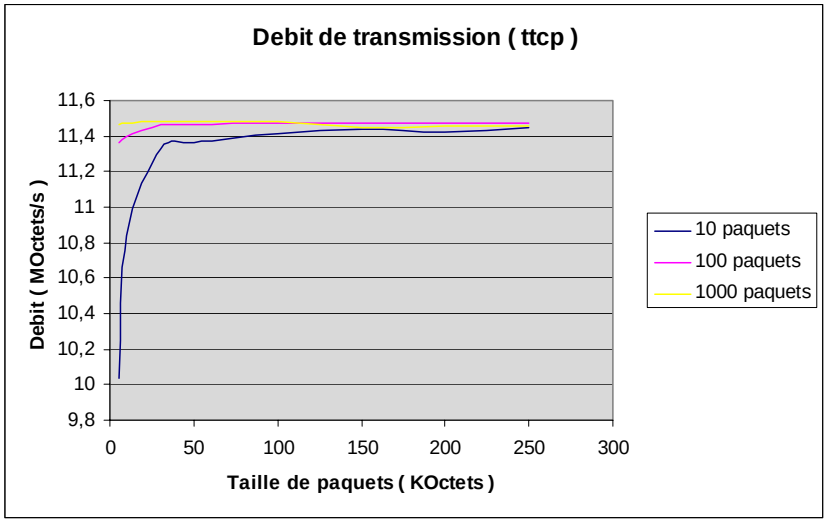
Lorsque la taille des tampons dépasse 20 KOctets, l'overhead devient négligeable et le débit des données plafonne à 90% de la bande passante du réseau Ethernet 100 utilisé.

Pour obtenir une bande passante maximale, il sera donc necessaire de grouper les données dont la taille est inférieure à 20 KOctets, de façon à utiliser des paquets de données d'une taille suffisante. Par contre, pour des données dont la taille dépasse les 20 KOctets, un regroupement en paquets de taille plus importante serait inutile, la bande passante maximale étant atteinte.

Mesures avec ttcp

Afin de vérifier l'efficacité de la communication avec XDAQ, il existe un logiciel destiné au test de communication sous TCP / IP : ttcp (Test TCP). Ce logiciel permet de réaliser des mesures de débit et de temps de transmission sur des paquets TCP. L'utilisateur peut spécifier le nombre de paquets à envoyer, la taille de chacun des paquets et le nombre des paquets à envoyer à la suite. Le test de la connexion se fait en simple sens : la version émettrice du logiciel envoi vers la version réceptrice une suite de paquets TCP, les paquets ne parcourant la liaison que dans un sens.

Pour pouvoir comparer avec les résultats obtenus sur XDAQ, le même type de mesures a été effectué : ttcp envoie des suites de paquets de données de tailles spécifiées et relève les bandes passantes obtenues pour chacune des tailles de paquets.



On remarque que la bande passante maximale utilisable (environ 11,5 MOctets soit 88% de la bande passante de la liaison Ethernet 100) est sensiblement la même que celle obtenue sous XDAQ, ce qui montre l'efficacité de la communication par XDAQ et le protocole I2O.

On remarque aussi une chute des performances pour des paquets de petites tailles. Là aussi, cette chute de performance s'explique par l'augmentation de l'overhead (la proportion de données du protocole et le temps processeur consacré au traitement du protocole augmentent). Cette chute de performance est tout particulièrement sensible lorsque le nombre de paquets envoyés est faible. Cela s'explique par la présence d'un tampon mémoire à l'émission. Les données ne sont réellement envoyées que lorsque le tampon mémoire TCP est plein. Pour de petite quantité de données, cela entraîne des retards à l'émission qui grève les performances.

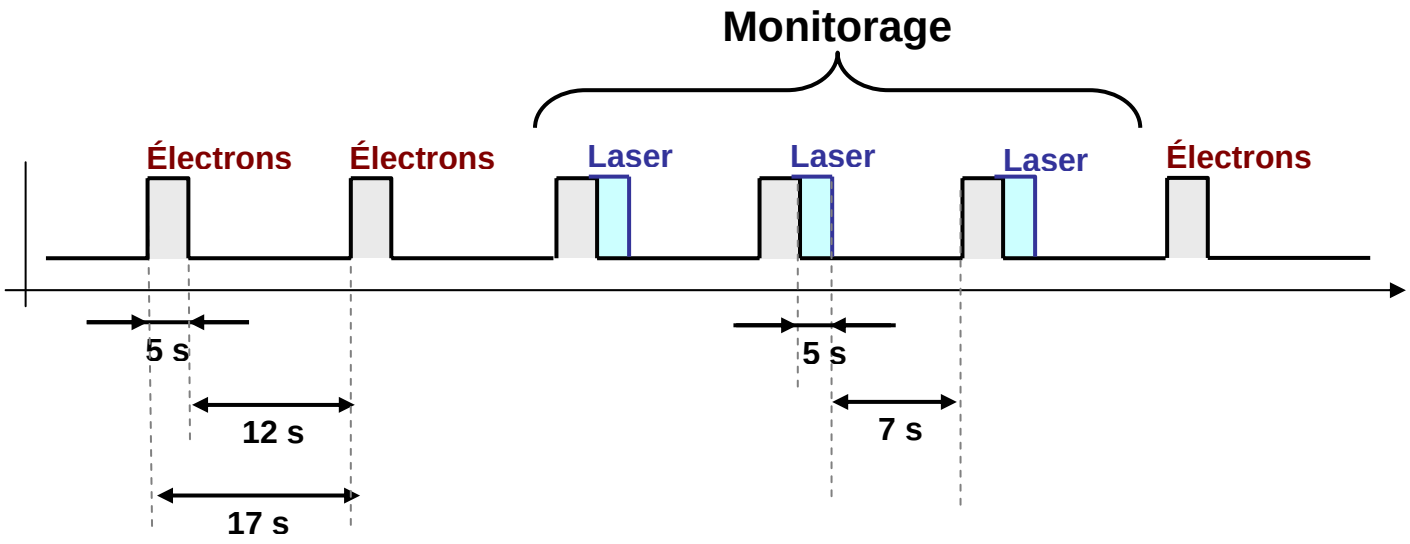
Sous linux, il existe une option : TCP_NO_DELAY, permettant de choisir si le système doit envoyer immédiatement les données passée par un programme ou s'il doit attendre quelque instant que d'autre données puissent arriver et augmenter la taille des paquets émis (et donc l'efficacité de la transmission). Dans le cas présent, cette options était désactivée, ce qui explique la chute de performance pour de petites chaînes (10 éléments) de petit paquets (<10 KOctets).

Simulation de la communication pour le fonctionnement en faisceau-test

Lors du prochain faisceau-test (au mois d'octobre prochain) il est prévu de calibrer les cristaux des supermodules déjà assemblés. Cette calibration s'effectue en envoyant un faisceau d'électrons (provenant de l'accélérateur SPS) sur chacun des cristaux. La structure du faisceau d'électrons est discontinue (voir la figure ci-dessous). Il s'agit d'une suite de paquets d'électrons de 5 secondes et séparés par des pauses de 12 secondes.

Le protocole de mesure est le suivant :

- Le faisceau-test est envoyé sur un cristal : toutes les 17 secondes, et durant 5 secondes, ce cristal recevra un paquet d'électrons (le faisceau d'électrons n'est pas continu). Les données correspondant à quatre tours (carré de 5 x 5 cristaux) entourant ce cristal seront lues par le système DCC. Une fois la mesure finie (à l'issue de la séquence des 3 expositions aux électrons), les données recueillies par la carte DCC seront envoyées, via XDAQ, sur le système DAQ.
- Toutes les 20 minutes, une mesure laser (monitorage) sera effectuée. Après les 5 secondes d'exposition aux électrons, les mesures électroniques sont abandonnées et une séquence de 5 secondes de calibrage laser est lancée. A l'issue des 5 secondes de calibrage, les données recueillies par la carte DCC seront envoyées simultanément sur le système DAQ et le système de monitorage. La séquence sera effectuée sur trois périodes du faisceau.



Pour les données des mesures électroniques (lors du faisceau d'électrons), en supposant que l'on prennent 10 échantillons, chaque séquence de mesure comporte 5 000 événements de 2 240 Octets chacun, soit 10.68 MOctets de données par mesure.

Pour les données de monitorage (lors du faisceau laser), chaque séquence de mesures comporte 500 événements de 20 160 Octets chacun, soit 9.61 MOctets de données par mesure.

Dans le cas des électrons, les données doivent arriver au DAQ en moins de 12 secondes. Dans le cas du monitorage, les données doivent arriver au DAQ et à la ferme de monitorage en moins de 5 secondes.

En première approximation, pour un réseau Ethernet 100, on peut espérer disposer d'environ 80% de la bande passante soit 10 MOctets/s. Ce qui veut dire que le transfert des données prendra environ 1 sec dans le cas de données de mesures (10.68 MOctets uniquement sur le DAQ) et 2 secondes dans le cas de données de monitorage (9.61 MOctets sur le DAQ et le monitorage). Un réseau Ethernet 100 doit donc suffire.

Afin de vérifier les performances, nous avons constitué un réseau simple avec les trois PCs. Deux serveurs tiendront lieu de DAQ et de Machine de monitorage et la machine de bureau tiendra lieu de serveur DCC. Les mesures ont été effectuées en construisant un réseau local avec seulement ces trois machines et un switch Ethernet 100BaseT

Le module de test XDAQ a été utilisé pour simuler l'envoi périodique des données 'laser' et 'électrons'. Dans ce cas de mesure, l'intégrité des données envoyées par le serveur sont vérifiées et libérée par les machines réceptrices.

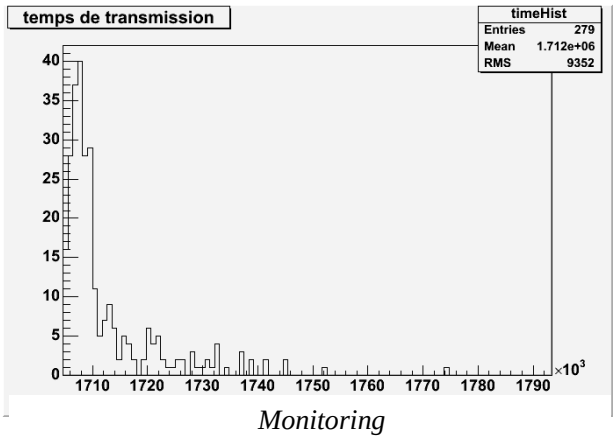
- La première série de mesures de performances correspond aux données de monitoring : toutes les 5 secondes (5 secondes suffisent pour le transfert des données), le serveur DCC envoie sur chacune des deux autres machines 9.61 MOctets de données sous la forme de chaînes de 500 tampons mémoire de 20 160 Octets chacun. Le temps d'envoi et la bande passante étaient relevés pour chaque envoi de données.
- La deuxième série de mesures de performance correspond aux données 'électrons' : toutes les 3 secondes, le serveur DCC envoyait sur le serveur DAQ 10.68 MOctets de données sous

la forme de 5 000 tampons mémoire de 2 240 Octets chacun. De même, le temps et le débit de transmission sont relevés.

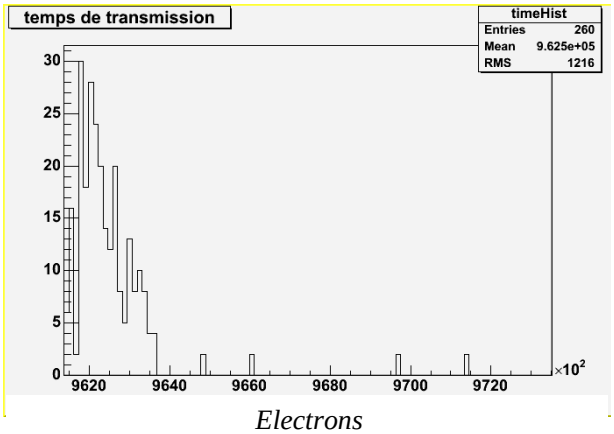
Afin de vérifier que la consommation de temps processeur n'influe par sur les performances de communication, un logiciel simple de consommation de temps processeur a été utilisé. Les deux séries de mesures présentées ci-dessus ont été refaites alors que le logiciel de charge processeur était en fonctionnement.

Monitoring				Monitoring (avec charge processeur)			
Temps de transmission (sec)		Débit de transmission (MOctets/s)		Temps de transmission (sec)		Débit de transmission (MOctets/s)	
moyen	1.7066	moyen	11.2103	moyen	1.70763	Moyen	11.1786

Electrons				Electrons (avec charge processeur)			
Temps de transmission (sec)		Débit de transmission (MOctets/s)		Temps de transmission (sec)		Débit de transmission (MOctets/s)	
moyen	0.958907	moyen	11.2709	moyen	0.958653	moyen	11.2849



Monitoring



Electrons

On remarque que le débit moyen est dans tous les cas supérieur au 80% attendus (il atteint même environ 90%). Les temps de transmission sont, eux, toujours inférieur à 2 secondes (cas du monitoring) et à 1 seconde (cas des mesures), ce qui confirme la faisabilité du système. Il y'a même une marge de manoeuvre assez importante.

La présence d'une charge processeur ne perturbe que peu les communications, elle provoque juste une légère baisse de performances, dans le cas des données de monitoring. La consommation processeur des applications qui fonctionneront sur les machines du test ne devrait donc pas perturber les transmissions de données.

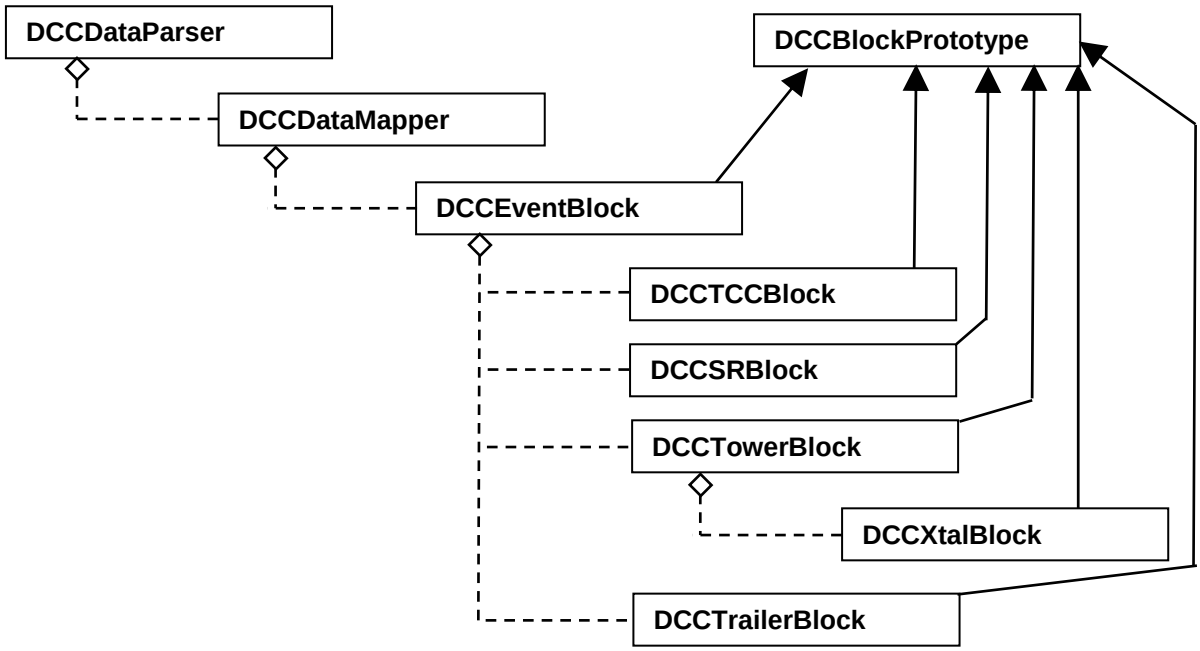
FORMAT DE COMMUNICATION

Les données récupérées par le serveur DCC ne sont pas retransmises telles quelles à la ferme de monitoring (et au système DAQ), elle sont recodée dans un format spécifique.

Le décodage de ce format fait appel à des classes implémentées par Nuno ALMEIDA.

Structure des classes de décodage

La hiérarchie des classes utilisées pour le décodage reprend la hiérarchie des données dans le Format de communication, en annexe I:



La classe DCCBlockPrototype est la classe mère des classes d'accès aux données. Elle contient les méthodes permettant d'accéder aux champs du bloque de données à partir de leur identifiant.

La classe DCCEventBlock représente un événement. Elle contient les données de trigger, de lecture sélective, et la liste des tours. Les classes DCCTCCBlock, DCCSRBlock, représentant respectivement les données relatives au trigger et les données relatives à la lecture sélective, ne seront pas utilisé dans le cadre du monitorage. La classe DCCTrailerBlock contient les données de fin d'événement (en complément des données d'entête).

La classe DCCTowerBlock représente une tour de cristaux. Elle contient la liste des cristaux, sous la forme d'instances de la classe DCCXtalBlock.

La construction d'une liste d'événements à partir des données se fait grâce à la classe DCCDataParser. Cette dernière permet d'analyser un fichier ou un tampon mémoire à la recherche de données représentant des événements. Lorsqu'un événement est trouvé, elle fait appel à la classe DCCDataMapper qui se charge d'instancier une classe DCCEventBlock et de la remplir avec les données. On obtient donc une liste d'événements, sous la forme de classes DCCEventBlock, permettant d'accéder facilement aux données.

Utilisation des classes de décodages

La récupération d'une liste d'événements, depuis un tampon mémoire, se fait en appelant la méthode parseBuffer de la classe DCCDataParser. On obtient alors un vecteur contenant tous les événements du tampon. Il suffit alors de faire une boucle sur le vecteur pour obtenir successivement chacun des événements. Il est possible de faire de même pour les niveaux de hiérarchie inférieurs (Tour et ensuite cristaux).

Pour chaque niveau de la hiérarchie, il est possible d'accéder aux champs de données par la méthode getDataField. Cette méthode prend en paramètre l'identifiant du champ (sous forme de texte) et retourne la valeur numérique du champ.

Dans le cas où une erreur se produit, une exception de type DCCException est propagée.

Exemple d'un programme simple permettant de décoder un fichier au format DCC, dans un fichier texte :

```
// header de Nuno
#include "DCCDataParser.hh"
#include "DCCEventBlock.hh"
#include "DCCException.hh"

// header standards
#include <iostream>
#include <iomanip>

// fonction principale du programme
int main( int argCount, char ** args )
{
  // besoin de deux parametres
  if( argCount < 3 )
  {
        cout << "usage : DataParser fileIn.txt fileOut.txt" << endl;
        return 1;
  }

  // interception des erreur de parsing
  try
  {
        // instanciation et parsing
        DCCDataParser myParser;
        myParser.parseFile( args[ 1 ] );


        // fichier de sortie
        fstream file( args[ 2 ], ios_base::out | ios_base::trunc );

        // liste d'événements et iterateur de parcours
        vector< DCCEventBlock * > &   dccDataBlocks = myParser.dccEvents();
        vector< DCCEventBlock * >::iterator it;

        // décodage et enregistrement
        ulong i(0);
        for( it = dccDataBlocks.begin(); it!= dccDataBlocks.end(); it++, i++)
        {
                (*it)->displayEvent( file );
            if( (*it)->eventInError() )
                {
                cout<<"\n "<<(*it)->eventErrorString()<<endl;
                }
        }

        // terminaison
        file.close();

  }
  catch ( DCCException &e)
  {
        // affichage de l'erreure
        cout <<"Parsing ended.. error : "<< e.what() << endl;
  }
```

```
    return 0;
}
```

MODULE DE COMMUNICATION

Le rôle du module de communication est de servir d'interface entre le serveur DCC (depuis le réseau local) et l'application de monitorage (AnaLas.exe, sur la même machine). Il doit recevoir les données relatives aux événements envoyées par le serveur DCC pour les mettre à disposition d'Analas.exe, et ce en utilisant l'environnement XDAQ.

Contraintes de fonctionnement

Le module de communication doit permettre la réception des événements en provenance du serveur DCC, pour les passer à l'application de monitorage.

L'application de monitorage ne pouvant pas systématiquement traiter les données au fur et à mesure de leur arrivée, le module doit être capable de conserver les données arrivées en attendant leur traitement.

Les données seront envoyées événement par événement, alors que l'application de monitorage les traite par run (paquet de 1500 événements). Le module devra donc être capable de regrouper les données relatives à un run pour les transmettre en une fois à l'application de monitorage.

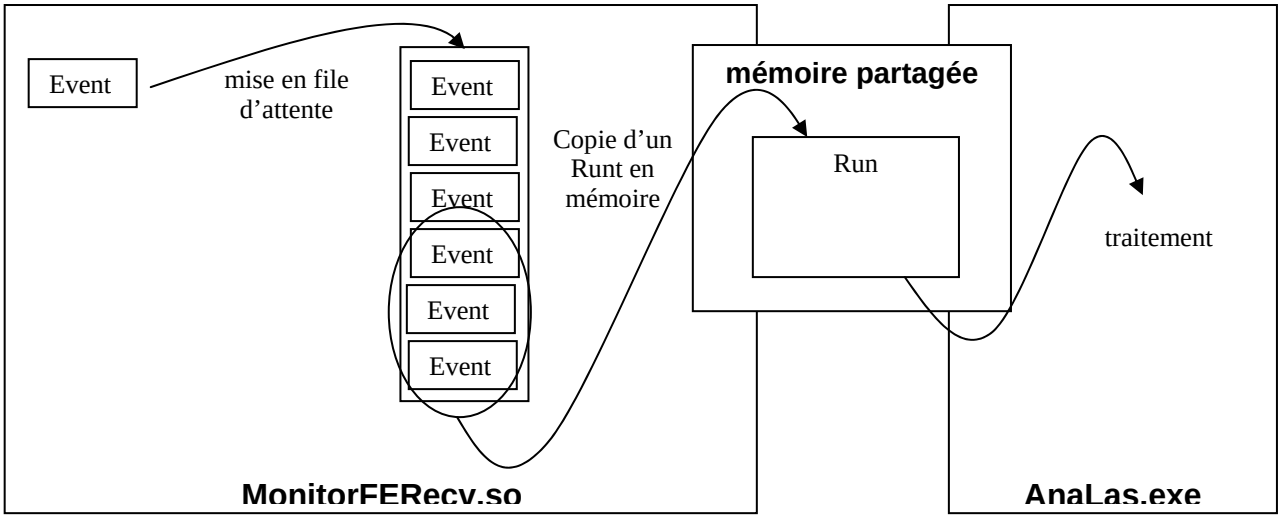
Le module devra être capable de repérer le début et la fin d'un run grâce aux informations contenues dans les messages I2O envoyés par le serveur DCC.

Principe de fonctionnement

De part le fonctionnement d'un module XDAQ, la réception des messages en provenance du serveur DCC se fait grâce à une fonction de callback appelée par l'exécutable XDAQ à chaque nouvelle arrivée d'un message. Cette fonction place alors l'événement reçu dans une file d'attente. Cette file d'attente permet de s'assurer que tous les événements sont bien enregistrés à leur arrivée, même si l'application de traitement n'est pas encore prête à les traiter.

Lorsque l'application de monitorage a fini le traitement des données précédemment arrivées, les données en file d'attente, et correspondant à une séquence complète, sont recopiées dans une zone de mémoire partagée. L'application de monitorage peut alors accéder à ces nouvelles données et commencer le traitement de la séquence suivante.

Si la réception des données est moins rapide que leur traitement, l'application de monitorage doit alors attendre l'arrivée d'une séquence complète avant de la traiter.

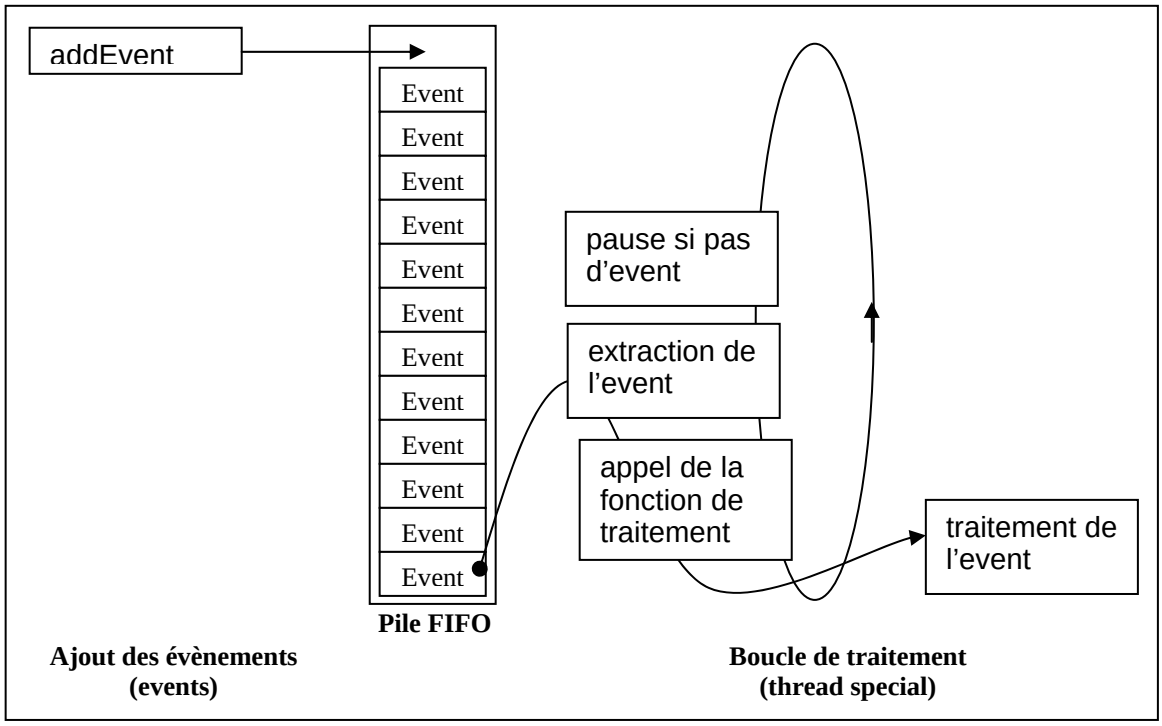


File d'attente

La file d'attente est gérée par une classe nommée ThreadProc. Cette classe contient une pile FIFO et des sémaphores. La fonction appelée pour le traitement des événements est spécifiée par l'utilisateur, au moment de l'instanciation de la classe. Les méthodes publiques de la classe permettent la gestion de la file d'attente : ajout d'un événement, vidage de la file d'attente, mise en pause des traitements...

Lors de son instanciation, la classe crée un nouveau thread qui utilisé pour la gestion de la pile FIFO. La fonction principale du thread contient une boucle de traitement. Cette boucle prend le premier élément de la pile FIFO et le passe à la fonction de traitement enregistré par l'utilisateur.

Si aucun élément n'est présent dans la pile FIFO, la boucle de traitement se met en pause, en attendant l'arrivée d'un nouvel événement. Lorsque qu'un nouvel événement est ajouté, la boucle de traitement quitte la pause et commence le traitement de l'événement. La mise en pause de la boucle se fait grâce à une condition d'arrêt, ce qui permet de ne pas consommer de temps processeur (comme ce serait le cas lors d'une attente avec scrutation).



Traitement d'un événement

Lorsque q'un événement arrive, le tampon mémoire contenant les données est enregistré dans une structure de type EventTask. Cette structure est ajoutée à la file d'attente.

La fonction de traitement prend ensuite chacun des événements d'un burst depuis la file d'attente pour les placer dans une seconde file d'attente (qui contiendra alors les données d'un seul burst). Une fois tous les événements d'un burst récupérés, les données de chaque événement sont copiée dans un segment de mémoire partagée. Pour savoir le numéro de l'événement en cours, et le nombre d'événements du burst, l'entête du message I2O envoyé par XDAQ contient une copie de ces informations. Il n'est donc pas nécessaire de faire appel aux classes de décodage dans le module de réception.

Voici le fichier d'entête définissant un message I2O :

```
/**
* @file I2OEventFrame.h
* @brief Structure d'un message I2O.
*
* @author Luc DURAND
*/

// protection anti inclusion multiple
#ifndef _I2OEVENTFRAME_
#define _I2OEVENTFRAME_

// inclusion des headers
#include "Globals.h"
#include "i2o.h"

// constantes
const int I2O_LASER_CODE = 0x0001;

/**
* @struct I2OEventFrame
* @brief structure d'un message I2O.
*/
struct I2OMsgFrame
```

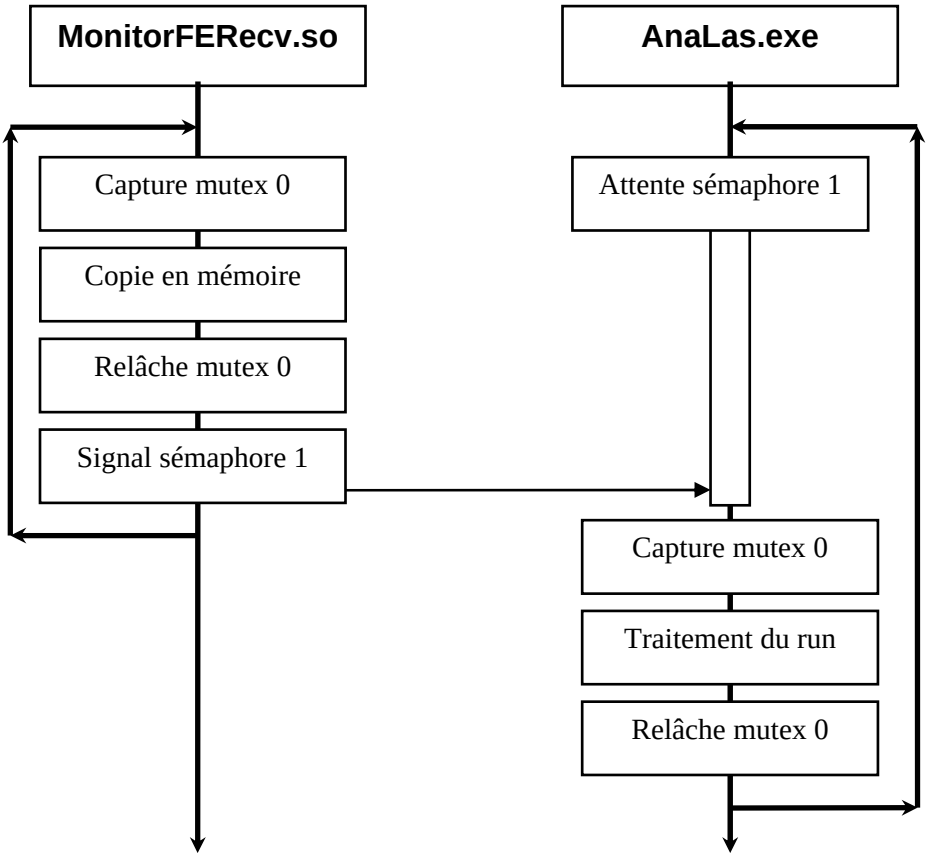
```
{
#if defined(LITTLE_ENDIAN__)
    // entete privee
    I2O_PRIVATE_MESSAGE_FRAME privateFrame;

    // reserve
    U32 reserved;
        U32 reserved0;
        U32 reserved1;

    // taille total du burst
        U32 totalLength;
    // taille de l'evenement
    U32 partLength;

    // reserve
    U32 reserved2;
        U32 reserved3;

    // nombre d'evenement du burst
        U16 totalNumEvt;
    // numero de l'evenement
        U16 currentEvt;

    // donnees DCC
    char data[ 1 ];
#elif defined(BIG_ENDIAN__)
    // entete privee
    I2O_PRIVATE_MESSAGE_FRAME privateFrame;

    // reserve
    U32 reserved;
        U32 reserved0;
        U32 reserved1;

        // taille total du burst
    U32 totalLength;
    // taille de l'evenement
        U32 partLength;

    // reserve
        U32 reserved2;
        U32 reserved3;

    // nombre d'evenement du burst
        U16 totalNumEvt;
    // numero de l'evenement
        U16 currentEvt;

    // donnees DCC
    char data[ 1 ];
#endif
};

// fin de protection anti inclusion multiple
#endif
```

L'application d'analyse (AnaLas.exe) peut alors récupérer les événements depuis le segment de mémoire partagé pour les traiter.

Synchronisation

Il n'y a pas de synchronisation entre le système de prise de mesure et l'application d'analyse. Le module de réception doit donc être capables d'assurer la bonne cohérence des données en mémoire partagées (toutes les données en mémoire viennent de la même séquence de mesure). L'application d'analyse ne doit pas pouvoir lire les données pendant qu'elle sont écrites par le module de réception, et inversement. Le module doit aussi être capable de prévenir l'application de traitement lorsqu'une nouvelle séquence de mesures vient d'arriver.

La synchronisation entre l'application de traitement et le module est réalisée grâce à deux sémaphores. Le premier sémaphore est utilisé en sémaphore d'exclusion mutuelle (mutex), pour protéger la mémoire partagée. Le second est utilisé comme une attente sur condition pour synchroniser l'application d'analyse sur la disponibilité des données.



Le premier sémaphore permet de protéger le segment de mémoire partagée. L'application d'analyse prend le mutex au début de son traitement et le relâche uniquement lorsque le traitement est complètement terminé. Le module XDAQ, lui, prend le mutex des qu'une séquence de mesures est prête à être recopié en mémoire et le relâche des la copie terminée. On s'assure ainsi que les données en mémoire ne peuvent être accédées que par un seul processus à la fois. Le module ne peut charger la mémoire partagée que lorsque l'analyse est terminée, et l'analyse ne peut être faite qu'une fois toutes les données d'un burst copiées.

Le second sémaphore permet de synchroniser le début de l'analyse avec l'arrivée des données, c'est à dire de faire en sorte que l'application d'analyse attende que les données soient disponibles sans consommer de temps processeur. Au début de son analyse (avant même d'accéder au segment de mémoire partagée), l'application de traitement attend sur le sémaphore. Le module XDAQ envoi un signal au sémaphore lorsqu'il a fini de copier les données d'une séquence de mesures en mémoire partagée, le début de l'analyse est synchronisé avec l'arrivée des données.

Si l'application d'analyse n'est pas lancée, le module de réception reste capable de fonctionner. Néanmoins, les données reçues alors que l'application d'analyse n'était pas lancée seront perdues.

Tests en situation

Dans le cadre du faisceau de test, le serveur DCC doit envoyer ses données à la fois sur le système DAQ et sur la ferme de monitorage, lorsqu'il s'agit de données provenant d'un burst laser.

De part les choix des différentes personnes travaillant sur chacune de ces parties, toutes n'utiliseront pas les mêmes technologies logicielles. Le serveur DCC et la ferme de monitoring utilise l'environnement XDAQ alors que le système DAQ utilise un simple socket TCP/IP. Afin de vérifier la bonne compatibilité entre ces différents logiciels et le fait qu'ils soient utilisables en conditions réelles, j'ai effectué une mission de quelques jours au CERN.

Le but de cette mission était donc de faire fonctionner ensemble, et dans des conditions les plus proches possibles de celles du faisceau de test, le serveur DCC, le daemon de réception du system DAQ et le module de réception de la ferme de monitoring. Etant donnée l'absence de données issues d'expériences au format DCC, les tests ont été fait avec des données générées par les outils de simulation de l'expérience CMS.

Les tests devaient être fait directement avec les machines du hall H4 (le hall où sont réalisés les faisceaux-tests) de façon à se rapprocher le plus possibles des conditions réelles d'utilisation. Malheureusement, il s'est avéré impossible de faire tourner le serveur DCC sur une des machines présentes. Nous avons néanmoins pu faire une partie des tests en utilisant une autre machine du CERN. Mais, cette machine n'étant pas directement sur le réseaux du hall H4, les débits de données entre le serveur et les deux logiciels de réception était plus faible (Ethernet 10 Mbits au lieu d'Ethernet 100 Mbits).

Même si les débits de communication sont restés plus faibles que désiré, ce test a tout de même permis de constater le bon fonctionnement de la communication entre les différents logiciels utilisés. Malgré l'utilisation de technologies de communication différentes, le système est fonctionnel. De plus, si l'on extrapole les débits observés lors de ce test à un réseau 100 Mbits, on constate que l'on atteint des débits suffisant pour l'utilisation en faisceau-test.

Remarques

Taille de mémoire partagée

La taille maximale de mémoire partagée qui puisse être allouée sur une machine donnée est fixée par le système. Dans le cas de linux, cette limite est fixée à 33 MOctets.

Dans le cadre des faisceaux test, la taille des données d'une séquence de mesures de 1500 événements, la taille des données approche les 32 MOctets. Si la taille des données devait augmenter, la taille limite d'un segment pourrait être dépassée, et il faudrait alors envisager de modifier le module de communication et l'application d'analyse pour qu'ils fonctionnent avec plusieurs segments de mémoire partagée. Un passage à deux segments (ou plus) demanderait une complexification de la synchronisation et surtout du décodage mémoire qui ferait chuter les performances.

ANALAS.EXE

L'application d'analyse des données laser, est AnaLas.exe. Jusqu'alors, l'analyse des données, en provenance de précédents faisceaux test, se faisait à partir des fichiers générés lors de ces faisceaux test. Cette méthode de lecture des données n'est pas assez performante pour être utilisée lors du fonctionnement final de CMS : beaucoup de temps est perdu dans les accès disques.

L'utilisation d'un segment de mémoire partagée permet de transmettre les données arrivant depuis le réseau à l'application d'analyse sans passer par le disque dur.

Modifications

Jusqu'alors, l'application faisait l'analyse d'un fichier de donnée sur le schéma suivant :

```
Initialisation

récupération des paramètres

ouverture du fichier de pedestals
ouverture du fichier de données
créations des fichiers de sortie

Pour passe = 1 à 3
{
  Pour chaque événement
  {
        si passe == 1
        {
              traitements sur les donnée
        }

        si passe == 2
        {
              traitements sur les donnée
        }

        si passe == 3
        {
              traitements sur les donnée
        }
}

fermeture des fichiers

terminaison
```

J'ai donc modifié le schéma de programme de façon à traiter en boucle tous les événements arrivant depuis le module de communication. Cette modification a consisté en l'ajout d'une boucle, et du système de synchronisation à base de sémaphores (et complémentaire de celui présent dans le module de réception).

J'ai aussi remplacé les anciennes méthodes d'accès aux données par les méthodes des classes de décodage. Le nouveau schéma de programmation est donc le suivant :

```
Initialisation
```

```
Initialisation mutex
Initialisation segment de mémoire partagée

récupération des paramètres

ouverture du fichier de pedestals
créations des fichiers de sortie

Boucle
{
  attente du mutex 1 (synchronisation)
  prise du mutex 0 (protection mémoire)

  analyse du tampon mémoire (décodage et création de la liste d'événements)

  Pour passe = 1 à 3
  {
        Pour chaque événement
        {
              si passe == 1
              {
                    traitements sur les donnée
              }

              si passe == 2
              {
                    traitements sur les donnée
              }

              si passe == 3
              {
                    traitements sur les donnée
              }
        }
  }

  relâchement mutex 0
}

fermeture des fichiers

terminaison
```

Afin de permettre à l'utilisateur de quitter « proprement » (sans fuite de mémoire…) l'application, j'ai aussi rajouté la gestion du signal UNIX SIGTERM correspondant à l'interruption d'un programme par la combinaison de touche « ctrl + c ». Lorsque ce signal est envoyé, la fonction terminator est appelée. Elle se charge des désallocations mémoire, de la fermeture des fichiers, du détachement du segment de mémoire et du détachement des sémaphores.

Benchmark

Pour tester et mesurer les performances de l'application d'analyse, une fois pourvue du module de communication et des classes de décodage, j'ai utilisé un module XDAQ me permettant d'envoyer des bursts de tailles variables (le nombre d'événements dans le burst est spécifié par l'utilisateur) sur l'application d'analyse.

Tous les événements des burst seront identiques. Il s'agit d'un événement généré par les outils de simulation du détecteur, et reprenant les caractéristiques d'un événement laser (pas de zéro suppression, ni de selective readout). La lecture des événements laser se faisant par demi super module d'alternativement 32 et 36 tours, et afin de simuler les conditions les plus contraignantes, les événements utilisés comporteront 36 tours. La taille d'un événement est de 21960 Octets.

Les mesures prises correspondent :

- Au temps de décodage du tampon mémoire.
- Au temps total de traitement d'un événement (décodage et analyse)
- Au temps absolue écoulé depuis l'arrivée du premier run.

A partir de ces mesures, on peut donc aussi obtenir le temps d'analyse d'un run et le temps de transition entre deux run.

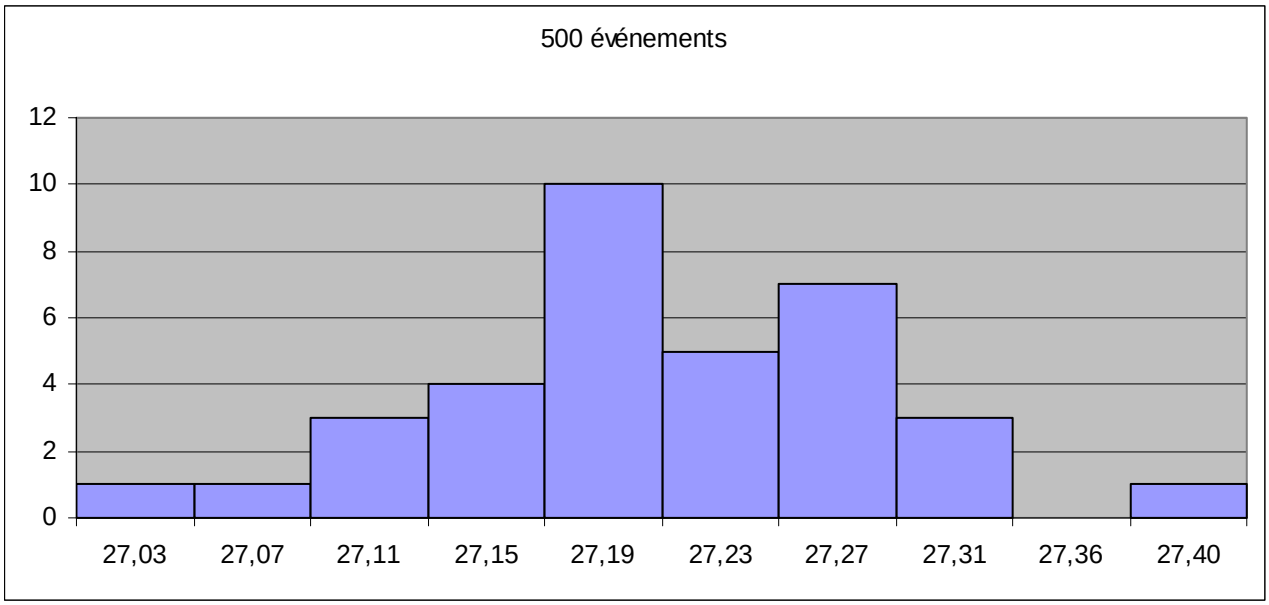
Afin de pouvoir quantifier les effets de la réception des données sur le temps de traitement, deux types de mesures on été faites :

- Réceptions et analyses séparées : les données ne sont envoyées qu'une fois l'analyse du run en cours terminée. Cette mesure permet de mesurer le temps d'analyse sans autre charge processeur.
- Réceptions et analyses simultanées : les données sont envoyées, même si le traitement du run en cours n'est pas terminé. Cette mesure permet d'évaluer l'impact de la réception de données sur le temps d'analyse.

J'ai donc envoyé sur le PC de monitoring des séquences de trois runs à 1500 ou 500 événements, séparées par des pauses correspondant à la durée de dix runs. Les traitements des deux premiers runs se déroulent donc alors qu'il y'a réception des données du run suivant. Le traitement du troisième run se déroule, par contre, sans qu'il n'y ait de réception de données.

Résultats pour 500 événements

	Traitement	**Décodage seul**
Temps moyen (s)	27.2	9.6
	Influence réception	**Chargement mémoire**
Temps moyen (s)	< 0.4	< 0.1



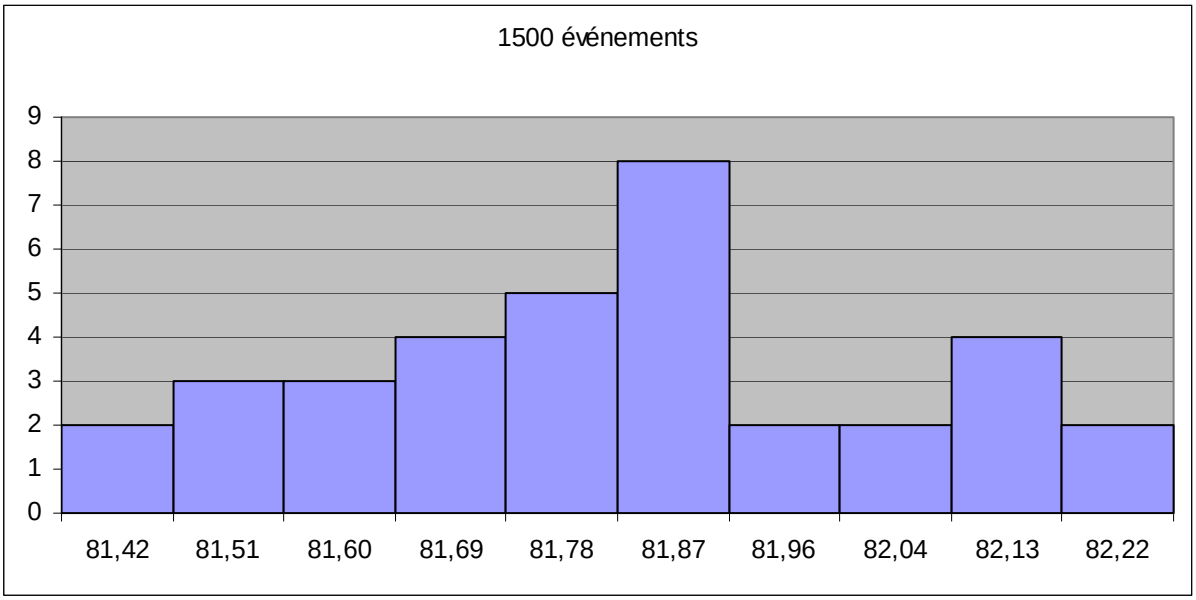
On peut constater que la réception des données n'influe pas de manière visible sur le temps de traitement. L'influence sur le temps de traitement des opérations de réception des données est négligable.

Le temps de décodage des données représente une part importante du temps total de traitement : en moyenne, 9.56 secondes sur les 27.19 secondes que dure le traitement soit 35.1%.

Le temps de transition, lorsque de nouvelles donnée sont disponibles (c'est-à-dire le temps consommé par les opérations sur les sémaphores et la copie des événements en mémoires partagée) est négligeable.

Résultats pour 1500 événements

	Traitement	**Décodage seul**
Temps moyen (s)	81.2	28.7
	Influence réception	**Chargement mémoire**
Temps moyen (s)	< 1	< 0.1



On constate, de la même façon que pour le test avec 500 événements, que la réception des données n'influe par sur le temps de traitement et que le temps de décodage des données représente 35.1% du temps de traitement total. Le temps de transition entre deux runs consécutifs est, là aussi, toujours négligeable.

En comparant avec le test à 500 événements, on constate que le temps de décodage et le temps de traitement total sont proportionnels au nombre d'événements dans le tampon mémoire.

CONCLUSION

J'avais choisi ce stage, en grande partie, pour mon intérêt pour la physique des particules. Au terme de ces six mois de stage, je m'estime satisfait : j'ai pu atteindre les objectifs de mon stage et découvrir une expérience telle que CMS.

Ce stage a été pour moi l'occasion de travailler avec des gens de nationalité différente. Malgré quelques difficultés de compréhension, j'ai toujours réussi à me faire comprendre de mes interlocuteurs. J'estime qu'il s'agit d'une bonne expérience de ce que pourra être mon travail à l'avenir.

J'ai aussi découvert un nouvel environnement de programmation : XDAQ. Bien que cet environnement soit spécifique au CERN et que j'ai peu de chance de le réutiliser dans l'avenir, je pense que c'est là aussi une expérience enrichissante.

Mon travail sur XDAQ a donc permis de valider le choix de XDAQ pour les communications réseau relatives au calorimètre électromagnétique. Il sera donc bien utilisé lors du prochain faisceau test.

Le module de communication que j'ai implémenté est pleinement fonctionnel et les tests de performance valide son utilisation e ne faisceaux-test. Il sera donc utilisé lors du prochain faisceau-test, en octobre 2004.

REMERCIEMENTS

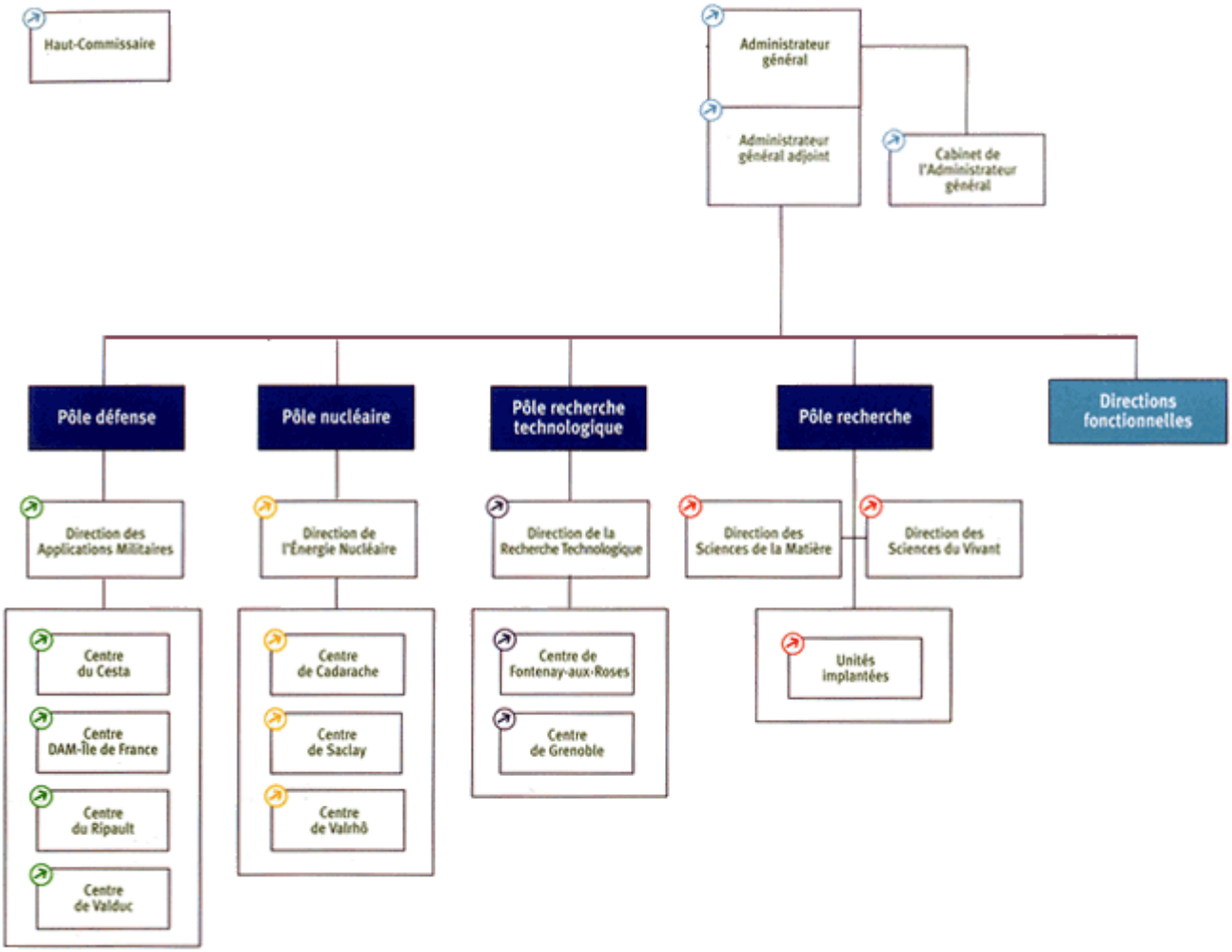
Je tiens à remercier toutes les personnes qui m'ont aidées et conseillées tout au long de ce stage, en particulier :

- Mon suiveur entreprise : M Philippe GRAS
- Mme Reyes ALEMANY FERNANDEZ
- M Shebli ANVAR
- M Jean BOUROTTE
- M Nuno Miguel GIRAO DE ALMEYDA
- Mme Marie-Claude LEMAIRE
- M Irakli MANDJAVIDZE
- M Bruno MANSOULIE
- M John RANDER
- M Patrice VERRECCHIA

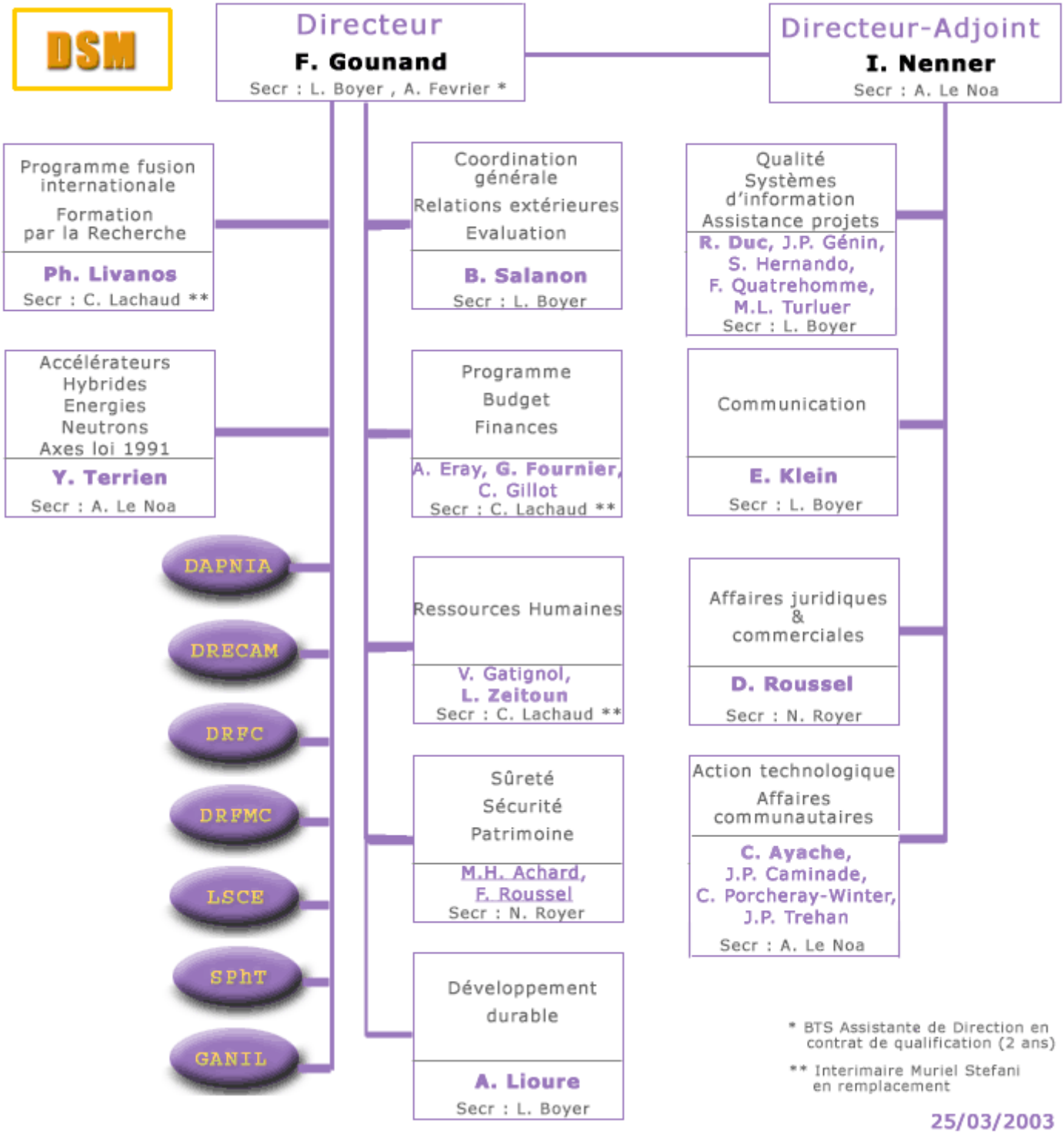
Je tiens aussi à remercier toutes les personnes du DAPNIA, du SPP, de l'équipe CMS et du SEDI qui m'on supportées durant ces six mois de stage.

ANNEXES

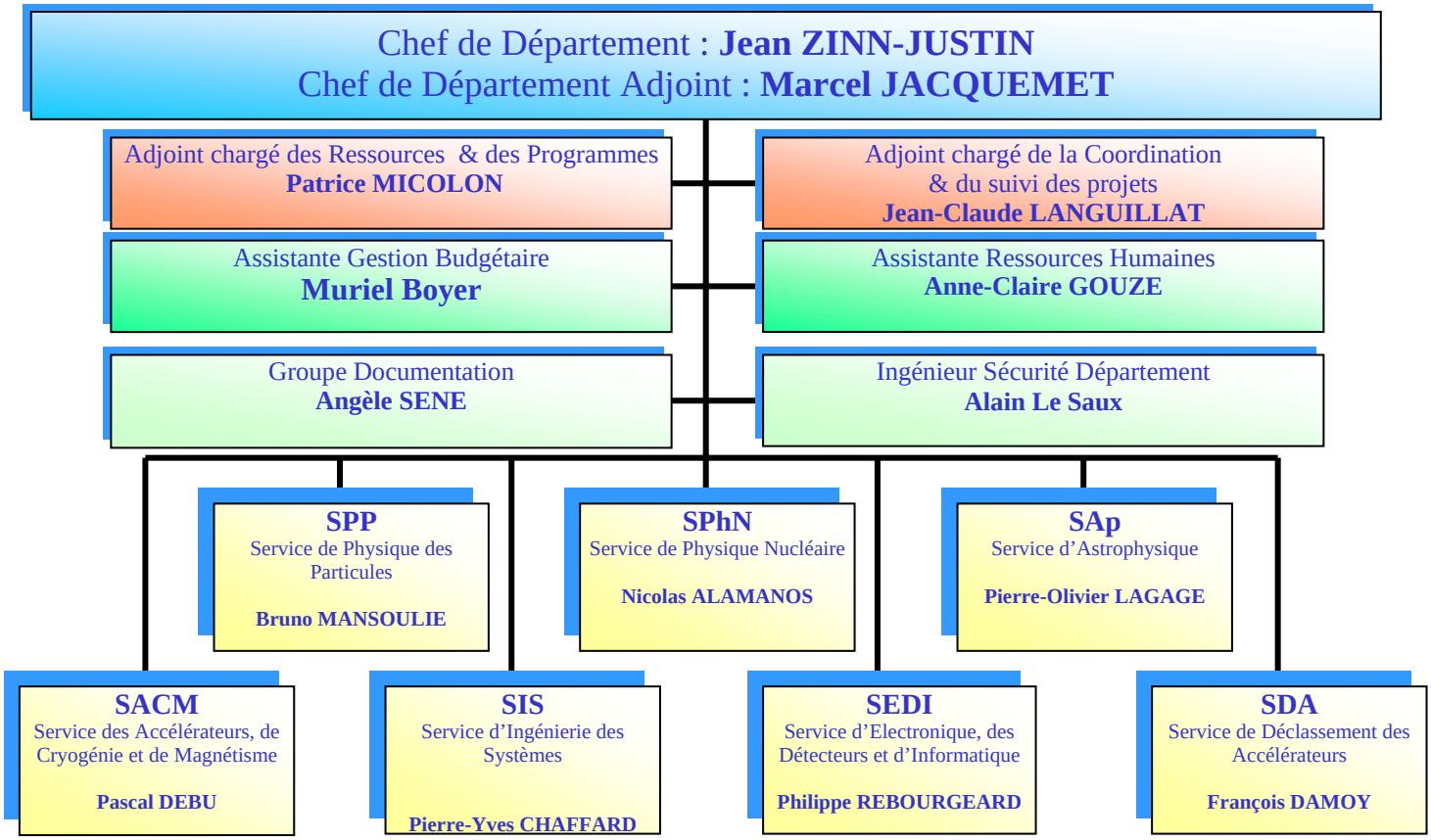
A. ORGANIGRAMME DU CEA



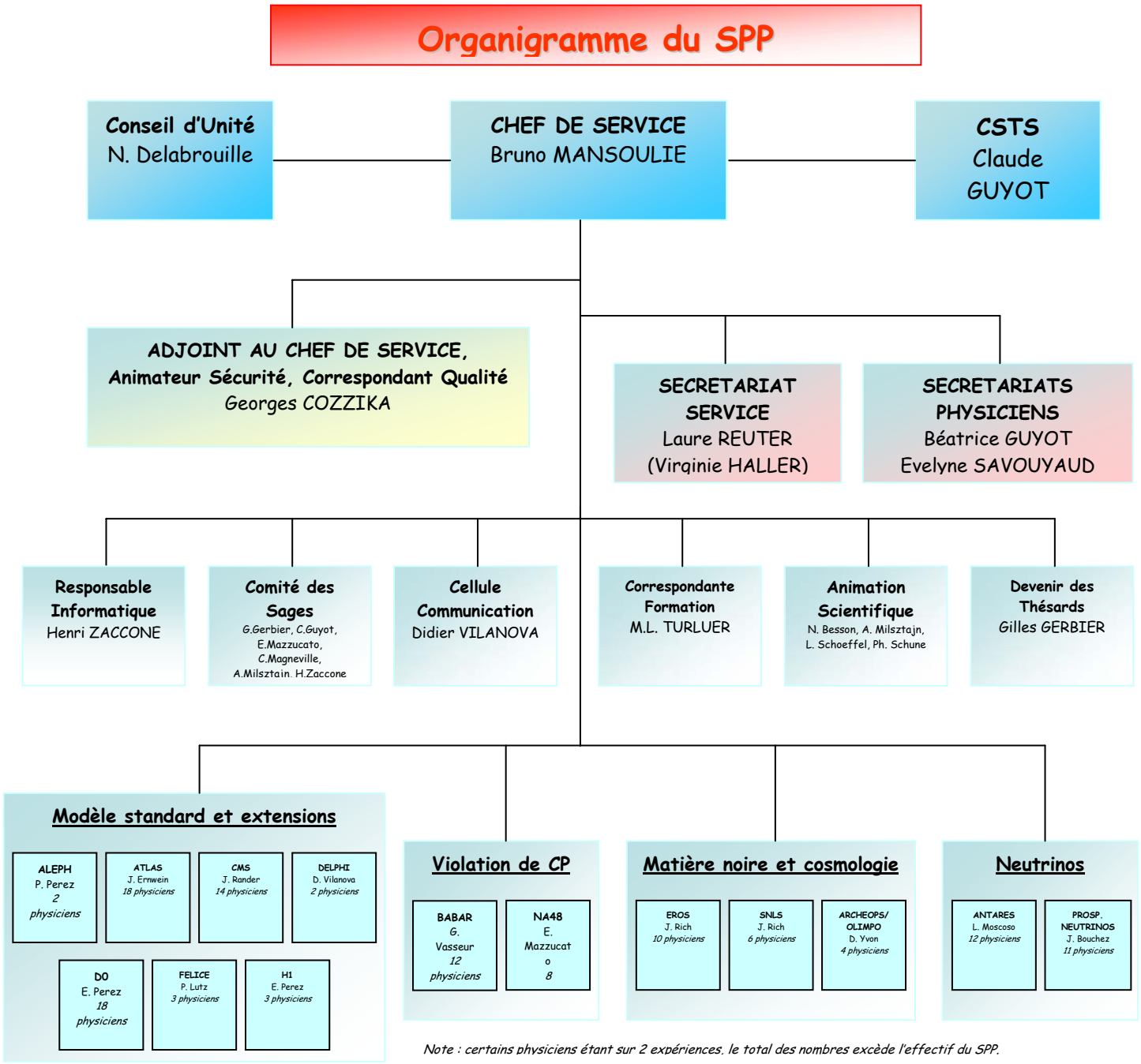
B. ORGANIGRAMME DE LA DSM



C. ORGANIGRAMME DU DAPNIA

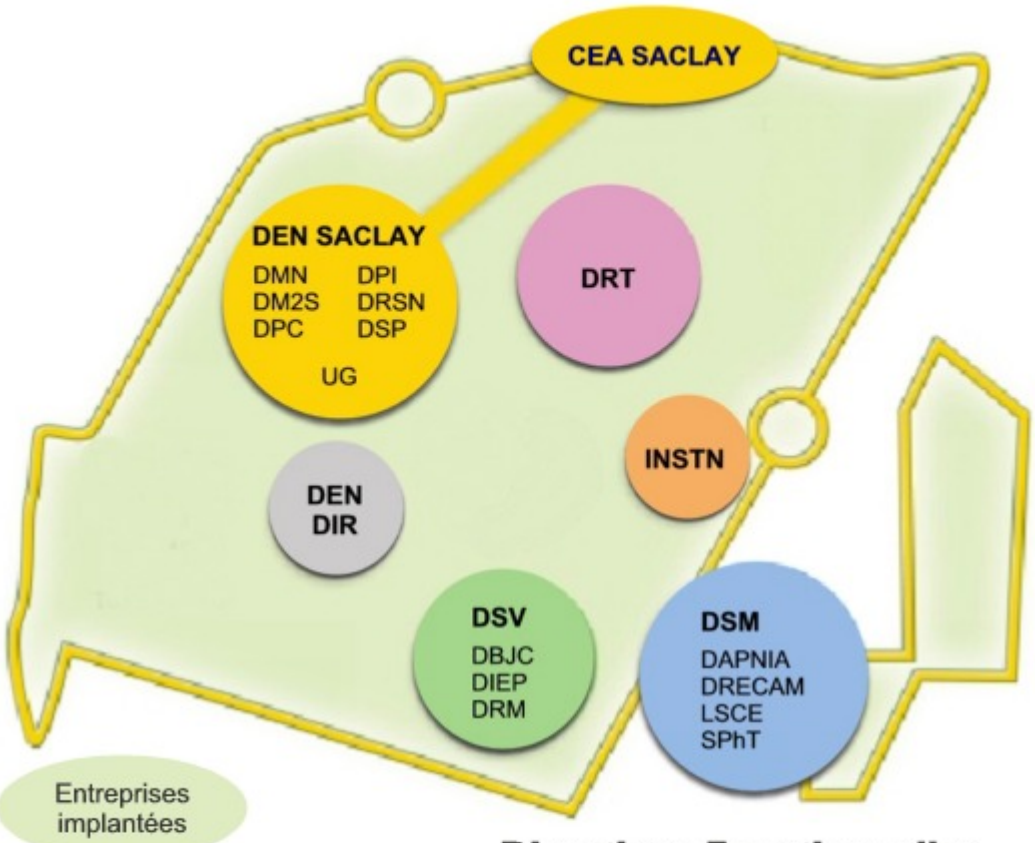


D. ORGANIGRAMME DU SPP



Note : certains physiciens étant sur 2 expériences, le total des nombres excède l'effectif du SPP.

E. ORGANISATION DU CENTRE CEA DE SACLAY



Entreprises implantées

SCHERING
CIS bio international
IRSN
ANDRA
euriso-top
TECHNICATOME

Directions Fonctionnelles

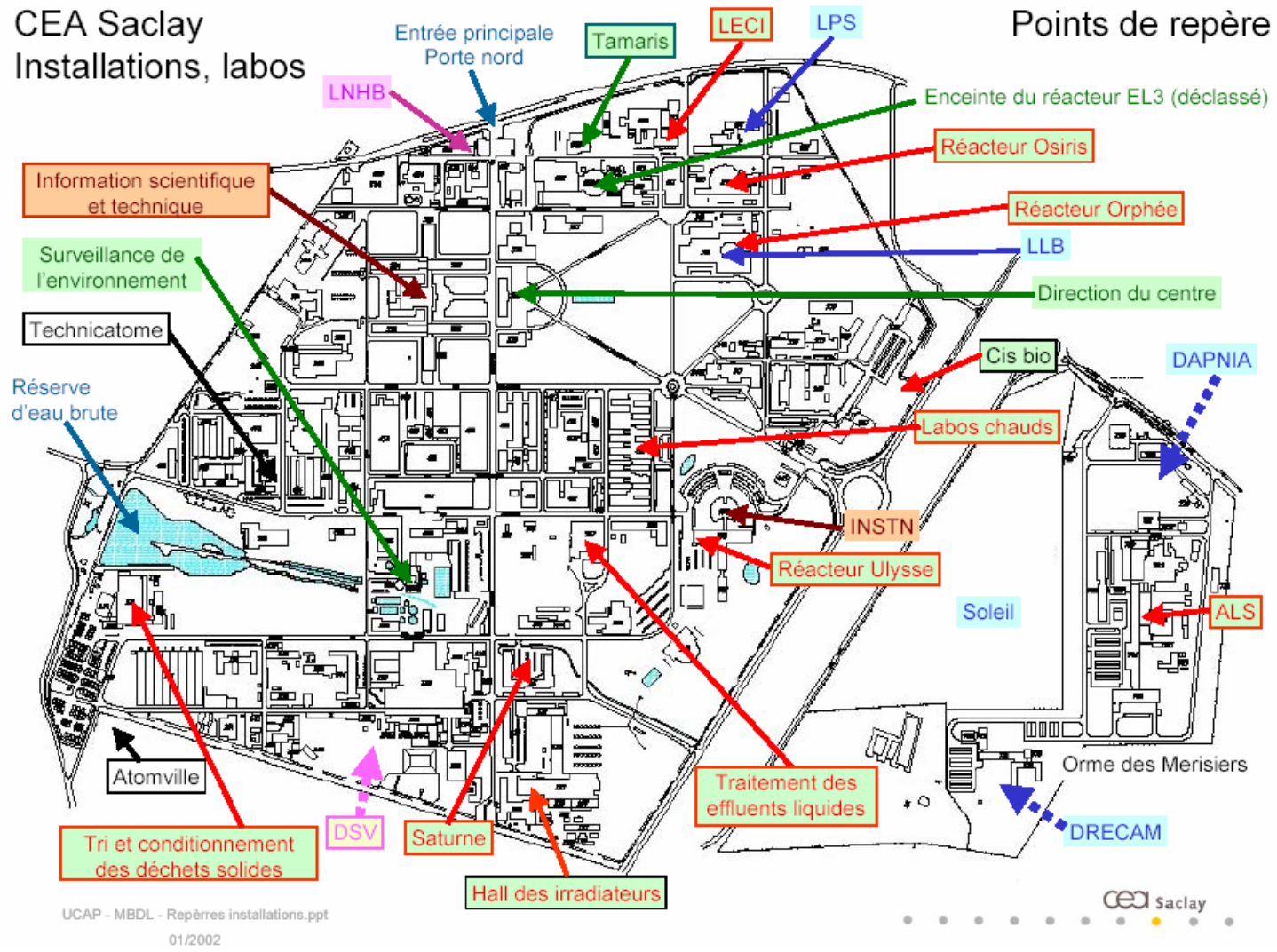
Pôle Maîtrise des Risques
- DCS (Direction Centrale de la Sécurité)
- DPSN (Direction de la Protection et de la Sûreté Nucléaire)
- DJC (Direction Juridique et du Contentieux)
- IGN (Inspection Générale et Nucléaire)

Pôle Gestions et Systèmes d'Information
- DAV (Direction Achats et Ventes)
- DF (Direction Financière)
- DSI (Direction des Systèmes d'Information)

Pôle Stratégie et Relations Exterieures
- DRI (Direction des Relations Internationales)

Pôle Ressources Humaines et Formation
- DRHRS (Direction des Ressources Humaines et des Relations Sociales)
- INSTN (Institut National des Sciences et Techniques Nucléaires)

F.PLAN DU CEA SACLAY



G. MESURES DE PERFORMANCES DE COMMUNICATIONS

Résultats pour 500 événements

temps (ms)			
traitement	décodage	total	transition
25799	8242	25799	0
27148	9511	53003	56
27116	9549	80177	58
27162	9559	251390	144051
27274	9531	278722	58
27025	9504	305804	57
27304	9606	472873	139765
27153	9583	500085	59
27084	9545	527226	57
27265	9611	693369	138878
27145	9571	720572	58
27191	9568	747820	57
27206	9599	914384	139358
27192	9621	941632	56
27196	9566	968886	58
27179	9605	1133430	137365
27257	9634	1160743	56
27192	9636	1187993	58
27212	9568	1356450	141245
27228	9651	1383736	58
27322	9618	1411114	56
27288	9629	1577525	139123
27252	9590	1604834	57
27195	9586	1632086	57
27207	9598	1798718	139425
27278	9619	1826053	57
27307	9601	1853417	57
27340	9646	2016465	135708
27204	9613	2043725	56
27183	9607	2070966	58
27285	9658	2239751	141500
27336	9674	2267142	55
27245	9641	2294444	57
27438	9663	2459638	137756
27246	9634	2486941	57
27291	9699	2514286	54
Moyennes			
27187	9564		56
			139470

Résultats pour 1500 événements

temps (ms)	

traitement	décodage	total	transition
77854	24892	77854	0
81833	28809	159779	92
81575	28729	241444	90
81868	28766	858172	534860
82314	28810	940578	92
81461	28739	1022126	87
82194	28851	1645096	540776
81790	29041	1726979	93
81417	28712	1808486	90
82197	28192	2421400	530717
81992	29068	2503479	87
81861	28824	2585432	92
82051	28856	3294487	627004
81516	28679	3376096	93
81732	28764	3457921	93
81934	28853	4064683	524828
81908	28896	4146673	82
81533	28755	4228296	90
81733	28699	4841840	531811
81916	28779	4923847	91
81929	28737	5005866	90
81882	28753	5622355	534607
81866	28820	5704320	99
81605	28755	5786016	91
81628	28730	6401936	534292
81827	28842	6483854	91
81952	28840	6565897	91
82073	28883	7182659	534689
81754	28816	7264503	90
81726	28878	7346321	92
81676	28782	7962189	534192
82150	28947	8044431	92
82018	28989	8126542	93
82277	28920	8741912	533093
82144	29069	8824146	90
81855	28885	8906091	90
Moyennes			
81751	28704		90
			541897

H. LEXIQUE DE LA PHYSIQUE DES PARTICULES

- **anti-particule** : une anti-particule possède la même masse que la particule correspondante, mais ses nombres quantiques sont opposés, en particulier la charge électrique
- **atome** : Un atome est la plus petite partie d'un corps pur. Il est formé d'électrons et de nucléons.

Le nom atome vient du grec et signifie "insécable". Maintenant, on sait que les atomes sont divisibles mais le nom est resté.

- **baryon** : Un baryon est un hadron composé de trois quarks, comme les nucléons.

Le nom baryon vient du grec et signifie "lourd". En effet, dans les années 1950, les trois catégories de particules connues étaient les leptons, les mésons et les baryons, correspondant aux mots grecs "léger", "moyen" et "lourd". Maintenant, on connaît un lepton plus lourd que certains baryons mais le nom est resté.

- **boson** : Un boson est une particule de spin entier, il obéit à la statistique de Bose Einstein. Les photons, les gluons, les W, le Z0 et le Higgs sont des bosons.
- **électrofaible** : La théorie électrofaible est une unification de QED et des interactions faibles à la base du modèle standard. La théorie électrofaible explique l'interaction faible comme une interaction entre fermions et W et prédit l'existence d'une interaction faible par courant neutre avec une interaction entre fermions et Z0.
- **électron** : Un électron est un lepton. C'est un des constituants de l'atome, avec les nucléons. Il a une charge électrique négative de -1,6 10-19C, qui est la charge électrique élémentaire (on dit donc que sa charge est -1). Sa masse est de 9,1 10-31kg, soit 0,511MeV.
- **électron-volt** : L'électron-volt (eV) est l'unité d'énergie utilisée en physique des particules : c'est l'énergie acquise par un électron soumis à un potentiel électrique de 1V. Ainsi, on a 1eV=1,6 10-19J, c'est donc une unité très faible. Les multiples sont le keV=103 eV, le MeV=106 eV, le GeV=109 eV...

Par la relation de la relativité restreinte E=mc2 (c=299 792,458km/s est la vitesse de la lumière dans le vide), on peut donc exprimer la masse en unités d'énergie. Ainsi, la masse de l'électron de 9,1 10-31kg correspond à une énergie de 0,511MeV.

- **fermion** : Un fermion est une particule de spin demi-entier, il obéit à la statistique de Fermi-Dirac. Les leptons et les quarks sont des fermions.
- **gluon** : Un gluon est un boson, c'est la particule qui transmet l'interaction forte.

Le nom gluon vient de "glue", car il permet de "coller" les quarks entre eux par l'interaction forte qui, comme son nom l'indique, est beaucoup plus forte que les autres interactions.

- **hadron** : Un hadron est une particule composite formée de quarks. Il y a deux sortes de hadrons : les baryons et les mésons.

Le nom hadron vient du grec et signifie "fort". En effet, les hadrons sont les particules qui sont sensibles à l'interaction forte, contrairement aux leptons.

- **Higgs** : Le boson de Higgs est la particule du modèle standard qui permet de donner une masse à toutes les autres particules. Il n'a pour l'instant pas été découvert.
- **lepton** : Un lepton est un fermion élémentaire insensible à l'interaction forte. Il existe six sortes de leptons : l'électron, le muon, le tau et les neutrinos électronique, muonique et tau.

Le nom lepton vient du grec et signifie "léger". En effet, dans les années 1950, les trois catégories de particules connues étaient les leptons, les mésons et les baryons, correspondant aux mots grecs "léger", "moyen" et "lourd". Maintenant, on connaît un lepton plus lourd que certains baryons mais le nom est resté.

- **méson** : Un méson est un hadron composé d'un quark et d'un anti-quark.

Le nom méson vient du grec et signifie "moyen". En effet, dans les années 1950, les trois catégories de particules connues étaient les leptons, les mésons et les baryons, correspondant aux mots grecs "léger", "moyen" et "lourd". Maintenant, on connaît un lepton plus lourd que certains baryons mais le nom est resté.

- **muon** : Un muon est un lepton. C'est la réplique de l'électron de la deuxième famille.
- **neutrino** : Un neutrino est un lepton. Il existe trois sortes de neutrinos : le neutrino électronique, le neutrino muonique et le neutrino tau.

Le nom neutrino signifie "petit neutre" par opposition au neutron dont la masse est élevée alors que le neutrino semble ne pas avoir de masse.

- **neutron** : Un neutron est un nucléon, il forme le noyau de l'atome avec les protons. Sa charge électrique est nulle.
- **nucléon** : Un nucléon est un baryon. C'est un des composants de l'atome avec les électrons (il forme le noyau atomique). Il y a deux sortes de nucléons : les protons et les neutrons.
- **photon** : Un photon est un boson, c'est la particule qui transmet l'interaction électromagnétique. Il est souvent noté gamma.

Le nom photon vient du grec et signifie "lumière". En effet, le photon transmet l'interaction électromagnétique, la lumière étant un exemple d'onde électromagnétique.

- **proton** : Un proton est un nucléon, il forme le noyau de l'atome avec les neutrons. Sa charge électrique est +1, l'inverse de celle de l'électron.

Le nom proton vient du grec et signifie "le premier", c'est en effet une des premières particules à avoir été identifiée.

- **QCD** : La QCD (Quantum ChromoDynamics ou chromodynamique quantique) est la théorie des interactions fortes dans le modèle standard. La QCD explique l'interaction forte comme une interaction entre quarks et gluons.
- **QED** : La QED (Quantum ElectroDynamics ou électrodynamique quantique) est la théorie des interactions électromagnétique, elle est inclue dans le modèle standard. La QED explique l'interaction électromagnétique comme une interaction entre particules électriquement chargées et photons.
- **quark** : Un quark est un fermion élémentaire sensible à l'interaction forte. Il forme les hadrons. Il existe six sortes de quarks : le quark up (u), le quark down (d), le quark étrange (s), le quark charmé (c), le quark beau (b) et le quark top (t).

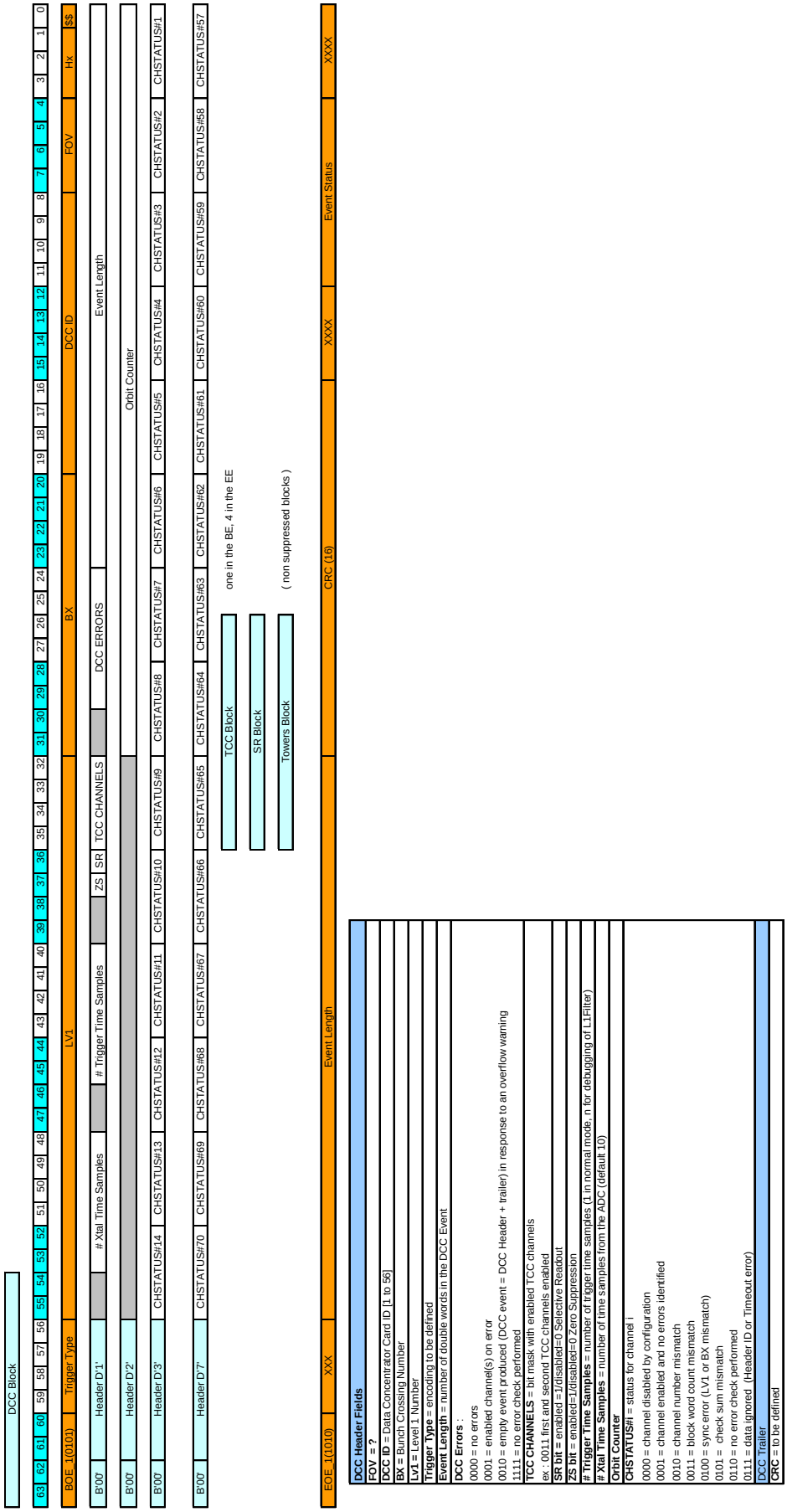
Le nom quark a été donné à ces particules par Gell-Mann et vient d'un poème de James Joyce Finnegan's Wake.

- **spin** : Le spin d'une particule est son moment angulaire intrinsèque. Le spin est une propriété quantique, il ne peut prendre que des valeurs entières ou demi-entières. Une particule de spin demi-entier est un fermion, une particule de spin entier est un boson.
- **tau** : Un tau est un lepton. C'est la réplique de l'électron de la troisième famille.
- **W** : Un W est un boson, c'est la particule qui transmet l'interaction faible.

  Le nom W vient de l'anglais et signifie "faible" (weak), car il transmet l'interaction faible qui, comme son nom l'indique, est beaucoup plus faible que les autres interactions.
- **Z0** : Un Z0 est un boson, c'est la particule qui transmet l'interaction faible par courant neutre.

  Le nom Z vient de "zéro" car sa charge électrique est nulle.

I. FORMAT DE COMMUNICATION



TCC Block — as from the tcc link

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
B'011'				TPG #1																TCC ID												B'011'	ERB	LV1																ERB	BX														
B'011'				TPG #5																TPG #4												B'011'				TPG #3														TPG #2														
B'011'																				TPG #68												B'011'				TPG #67														TPG #66														

TCC Fields
BX = Bunch Crossing ID (LSB)
LV1 = Level 1 Number (LSB)
ERB = Error Bit. It is on if there is a mismatch between BX or LV1 LSBs and TTC info
TCCID = Trigger Concentrator Card ID
TPG #i = trigger primitive of tower i

SR Block

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
B'100'		SR#4			SR#3			SR#2			SR#1									SRP ID												B'100'	ERB	LV1																ERB	BX														
B'100'		SR#68			SR#67			SR#66			SR#65									SR#64			SR#63			SR#62			SR#61			B'100'		SR#60			SR#59			SR#58			SR#57									SR#56			SR#55			SR#54			SR#53		

SRP Fields
BX = Bunch Crossing ID (LSB)
LV1 = Level 1 Number (LSB)
ERB = Error Bit. It is on if there is a mismatch between BX or LV1 LSBs and TTC info
SRP ID = Selective Readout Processor ID
SR#i = Selective Readout Flag for tower/SuperCrystal i

Tower Block

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
B'11'				Tower Block Length																Tower/Super Crystal ID												B'11'		ERB	LV1															ERB	BX														
B '11'	G	ADC Value					G	ADC Value																								B'11'	G	ADC Value																M	Xtal ID			Strip ID									
B '11'	G	ADC Value					G	ADC Value																								B '11'	G	ADC Value																G	ADC Value														
B '11'	G	ADC Value					G	ADC Value																								B '11'	G	ADC Value																G	ADC Value														
B '11'	G	ADC Value					G	ADC Value																								B'11'	G	ADC Value																M	Xtal ID			Strip ID									
B '11'	G	ADC Value					G	ADC Value																								B '11'	G	ADC Value																G	ADC Value														
B '11'	G	ADC Value					G	ADC Value																								B '11'	G	ADC Value																G	ADC Value														

TOWER BLOCK
BX = Bunch Crossing Number (LSB)
LV1 = Level 1 Number (LSB)
ERB = Error Bit. It is on if there is a mismatch between BX or LV1 LSBs and TTC info
Tower/Super Crystal ID [1…68]
Tower Block Length = number of double words in the block
G = gain
ADC value
M = monitoring bit
Strip ID = strip ID inside the tower [1…5]
Xtal ID = xtal ID inside the strip [1…5]