

L'Université de Genève en collaboration avec
le Centre Européen de Recherche Nucléaire
1999

Projet du diplôme :

**Conception et implémentation d'une interface
graphique utilisateur pour le prototype DAQ
d'Atlas en expérimentation au CERN**

Auteur : Sidiki CAMARA

Superviseurs : Dr David SQUIRE & Pr. Alan CLARK

Co-superviseur : Dr Lorenzo Moneta

Remerciement

Je remercie le Prof. Alan Clark de m'avoir fait confiance en m'acceptant dans son équipe de DAQ au CERN, je remercie le Dr Lorenzo MONETA du groupe DAQ d'Atlas au CERN pour sa disponibilité en m'assistant le long de tout le développement de ce projet, je remercie Dr Mihai CAPRINI et Dr Robert JONES tous les deux du groupe DAQ au CERN pour leur collaboration, je remercie également Dr David SQUIRE pour avoir accepté d'être mon superviseur au CERN dans ce travail, pour sa disponibilité en m'assistant pendant tout le long du développement et la rédaction de ce projet. Je remercie aussi tout le département de physique nucléaire de l'université de Genève pour avoir bien voulu mettre à ma disposition les ressources nécessaires pour l'élaboration de ce travail. En fin je remercie le centre universitaire d'informatique de l'université de Genève pour son effort dans mon apprentissage de l'informatique.

Sommaires

<i>Préface</i>	6
1. DESCRIPTION DU PROJET	8
1.1 LES EXIGENCES ET CONTRAINTES.....	8
1.1.1 <i>Les exigences et contraintes générales</i>	8
1.1.2 <i>Les exigences et contraintes d'affichage</i>	8
1.1.3 <i>Les exigences et contraintes du contrôle</i>	9
1.2 INTERACTION AVEC LES AUTRES COMPOSANTS BACK-END.....	9
2. LE PROTOTYPE-1 DU SYSTÈME D'ACQUISITION DE DONNÉES ET FILTRE D'ÉVÉNEMENTS (DAQ/EF) D'ATLAS	10
2.1 ABSTRACTION.....	10
2.2 INTRODUCTION.....	10
2.3 LE SYSTÈME TRIGGER/DAQ D'ATLAS.....	11
2.4 LE SYSTÈME FRONT-END D'ATLAS DAQ.....	12
2.5 APERÇU SUR LES FONCTIONNALITÉS DE FLUX DE DONNÉES.....	12
3. LE SOUS-SYSTÈME BACK-END DU SYSTÈME DAQ	15
3.1 GÉNÉRALITÉ.....	15
3.1.1 <i>Modularité et fiabilité de Back-end</i>	16
3.1.2 <i>Gestion des Partitions</i>	16
3.2 LES DIFFÉRENTS COMPOSANTS QUI COMPOSENT LE SYSTÈME BACK-END.....	16
3.2.1 <i>Le Composant «Configuration Database»</i>	17
3.2.2 <i>Le Composant «Process Manager System» (PMG)</i>	18
3.2.3 <i>Le Composant «Information Service» (IS)</i>	20
3.2.3.1 <i>Relation entre le MRS et l'IS</i>	20
3.2.3.2 <i>Structure de l'Information Service</i>	21
3.2.3.3 <i>Configuration Multi-serveur</i>	21
3.2.3.4 <i>Partitionnement DAQ</i>	22
3.2.3.5 <i>Exemple de fonctionnement d'IS</i>	23
3.2.3.6 <i>Les Sources d'informations et les types d'informations</i>	23
3.2.3.7 <i>Le Format d'Informations</i>	24
3.2.4 <i>Le Composant «Run Control» (RC)</i>	24
3.2.4.1 <i>Structure de Run Control</i>	25
3.2.4.2 <i>Le CHSM générique de contrôleur DAQ</i>	26
3.2.4.3 <i>Récupération d'erreurs</i>	29
3.2.4.4 <i>Les Communications du DAQ Superviseur</i>	32
3.2.4.5 <i>Les éléments du composant Run Control</i>	33
3.2.4.6 <i>Interaction entre les contrôleurs</i>	33
3.2.4.7 <i>Signalisation d'erreurs dans l'hierarchie du Run Control</i>	34
3.2.5 <i>Message Reporting System (MRS)</i>	35
3.2.5.1 <i>Vu d'ensemble</i>	35
3.2.5.2 <i>Structure du MRS</i>	36
3.2.5.3 <i>Définition de messages</i>	37
3.2.5.4 <i>Critère de souscription</i>	38
3.2.5.5 <i>Assemblage de message</i>	38
3.2.5.6 <i>Filtrage des messages</i>	39
3.2.5.7 <i>Les fonctions de contrôle du fichier login</i>	39
3.3 CONCLUSION.....	39
4. L'INTERFACE GRAPHIQUE GÉNÉRALE UTILISATEUR OU GRAPHIC USER INTERFACE (GUI)	41
4.1 VU D'ENSEMBLE.....	41
4.2 ANALYSE ET MODÉLISATION DU COMPOSANT « GUI_RC INTERFACE ».....	42
4.2.1 <i>Use case d'un scénario de lookup pour rc_handler et un is_handler</i>	42
4.2.2 <i>Use case pour un scénario d'envoi de commandes et de réception de nouvelles informations</i>	45
4.2.3 <i>Event Trace de Use Case "Look_rc_handler"</i>	47
4.2.4 <i>Event Trace pour le Use Case "Update_display"</i>	49

4.2.5	Event Summary diagramme pour la classe Rc_displayfinal	51
4.2.6	Modèle d'objet du composant "GUI_RC Interface"	53
4.2.7	Modèle d'objets du composant GUI_MRS Interface.....	57
4.2.8	Contexte de GUI intégrant les composants GUI_RC Interface et GUI_MRS Interface.....	58
4.3	LE COMPOSANT «GUI_RC INTERFACE»	60
4.3.1	L'Interface Graphique Utilisateur «RCCommands»	60
4.3.2	L'Interface Graphique des statuts des contrôleurs «RcStatusDisplay»	61
4.3.3	L'interface Graphique «RcErrorCommands».....	63
4.3.4	Le sous-composant «GUI_RC_IS Communication»	64
4.4	LE COMPOSANT «GUI_MRS INTERFACE»	65
4.4.1	L'interface graphique des informations formatées du MRS «MrsTable»	65
4.4.2	L'Interface graphique d'envoi de commandes MRS «MrsCmdGui».....	66
4.4.3	L'interface «Mrs logfile Display»	67
4.5	CONCLUSION.....	68
5.	L'INTÉGRATION DE GUI.....	69
5.1	RÉSULTAT DE L'INTÉGRATION DU SOUS-COMPOSANT «RCCommandes» DANS GUI	69
5.2	RÉSULTAT DE L'INTÉGRATION DES SOUS-COMPOSANT « RCSTATUSDISPLAY » ET «RCERRORCOMMANDS» DANS LE GUI	70
5.3	RÉSULTAT DE L'INTÉGRATION DU SOUS-COMPOSANT «MRSTABLE» DANS LE GUI	71
5.4	RÉSULTAT DE L'INTÉGRATION DU SOUS-COMPOSANT « MRSCMDGUI » DANS GUI	72
5.5	POURQUOI JAVA ?	72
5.6	CONCLUSION.....	73
ANNEXE 1.....		74
1.	LES DIFFÉRENTS NIVEAUX DES TRIGGERS.....	74
1.1	LE NIVEAU-1 DE TRIGGERS (LVL1).....	74
1.2	NIVEAU-2 TRIGGER.....	75
1.2.1	Sélection des événements (Event selection).....	76
1.2.2	Le mode d'opération et de partition.....	76
1.2.3	Description fonctionnelle des composants du niveau-2	77
1.2.3.1	Le Complexe ROB.....	77
1.2.3.2	Le Superviseur et Le ROI Builder.....	77
1.2.3.3	Le Réseau.....	77
1.2.3.4	L'environnement de traitement	78
1.3	PROBLÈMES ET CONTRAINTES	78
1.4	LES PARAMÈTRES D'ENTRÉE.....	80
1.4.1	Les informations de LVL1	81
1.4.2	Les informations fournies par LVL1 à LVL2.....	81
1.5	PROCESSUS DE SÉLECTION ET STRATÉGIE	82
1.5.1	Extraction de «type» Muons.....	83
1.5.2	Extraction de «type» Calorimètre	83
1.5.3	L'extraction de «type» e/γ.....	84
1.5.4	L'extraction de «type» Tau/hadron.....	84
1.5.5	L'extraction de «type» jet.....	84
1.5.6	Calcul de E_T^{miss} et $\sum E_T$	85
1.5.7	Extraction de «type» des détecteur d'intérieurs (TRT, SCT et Pixel)	85
1.6	CONCLUSION.....	85

Table des abréviations

Back-end

regroupe les composants logiciels du système d'acquisition de données.

CERN

European Laboratory for Particle Physics (Centre Européen de Recherche Nucléaire).

DAQ

Data Acquisition Query (le système d'acquisition des données).

DAQ/EF-1

A functionally complete pre-prototype DAQ and Event Filter system (le prototype-1 de DAQ et le filtre d'événement).

EB

Event Builder (le constructeur d'événement).

EF

Event Filter (le filtre d'événement).

Front-end

Shorthand for Front-end Electronics.

Front-end Electronics

The detector sub-systems which generate and send trigger data to the Level-1 Trigger System and event data to their RODs for transmission to the data acquisition system (les sous-systèmes de détecteurs qui génèrent et envoient les données au niveau-1 du système triggers et les événements de données à leurs RODs pour la transmission du système d'acquisition de données).

GUI

Graphic User Interface (Interface Graphique Utilisateur).

ILU

Inter-Language Unification.

IS

Information Service System (un composant logiciel du Back-end, le Système de Service d'Information dans le DAQ).

LHC

Large Hadron Collider.

LVL1

Level-1 Trigger System (le niveau-1 du système des triggers).

LVL2

Level-2 Trigger System (le niveau-2 du système des triggers).

MRS

Message Reporting System (un composant software du Back-end, le système de transport et de distribution des messages d'erreurs dans le DAQ).

PMG

Process Manager (un composant software du Back-end, le système de gestion des processus le DAQ).

RC (ou RunCtrl)

Run Control System (composant software du Back-end, le Système Run Control).

ROB (Readout Buffer)

A standard module which receives data from the RODs via standard Readout Links, passes on request a subset of the data to the Level-2 trigger, and buffers the data until a Level-2 Trigger decision has been reached whereupon, for accepted events, it transmits the data to the Event Filter. (Un module de buffers standards qui reçoit des données de Readout Links, les conserve pendant toute la durée de la prise de décision du Level-2 afin d'envoyer sur demande un sousensemble sélectionné de ces données au niveau-2 des triggers).

ROD (Readout Driver)

The last element in the Readout chain that is still considered part of the Front-end electronics. This module collects one or more data streams from detector elements and merges them into a single stream which is fed via a standard Readout Link into a ROB. (Le dernier élément de la chaîne de Readout faisant partie de l'électronique du Front-end. Son rôle est de collecter un ou plusieurs flux de données et les rassemble en un seul flux de données).

ROI (Region of Interest)

A geographical region of the experiment, limited in μ and ϕ , identified by the Level-1 Trigger System as containing candidates for Level-2 Trigger objects requiring further computation. Their data will be further analysed by the Level-2 Trigger System to decide if the event is to be processed further. In the case of β -physics triggers at low luminosity, some ROIs may be defined internally within the Level-2 Trigger. (Ce sont des régions où des données intéressantes ont été détectées pour le niveau-2 des triggers).

SCT

Semi-Conductor Tracking detector.

TRT

Transition-Radiation Tracking detector.

Table des symboles

EM	Electromagnétique.
E_T	Energie transverse.
P_T	Impulsion transverse.
μ	Rapidité d'un événement.
ϕ	Angle azimutal d'un événement.
$\mu-\phi$	rapidité des événement en fonction de l'angle azimutal.
$\Delta\eta$ (ou $d\eta$)	Variation instantanée de η .
$\Delta\phi$ (ou $d\phi$)	Variation instantanée de ϕ .
$\Delta\eta \times \Delta\phi$ (ou $d\eta \times d\phi$)	Les coordonnées $(\Delta\eta, \Delta\phi)$.
E_T^{miss}	Energie transverse manquante.
$\sum E_T$	Somme totale des énergies transverses.
High- P_T	Les hautes impulsions transverses.
Low- P_T	Les faibles impulsions transverses.

Préface

Le groupe d'Acquisition de données d'Atlas a produit un prototype de système représentant une architecture puissante d'acquisition de données (Le DAQ) avec l'intention d'évaluer les technologies et architectures futures, candidates pour le système final qui pourraient être utilisées pour mener à bien l'expérience d'Atlas concernant le Large Hadrons Collider (Le LHC) accélérateur au CERN.

Le back-end software permet de configurer, intégrer tous les composants de DAQ et contrôler le DAQ, tout en se comportant comme le moniteur du DAQ. Il est responsable de garder et de faire travailler ensemble les différents sous-systèmes du DAQ. Ce software inclus également des programmes de base pour contrôler et gérer l'acquisition des données et le statut du système DAQ; permet également l'échange d'information entre les différents composants softwares et rapporter des messages d'erreurs. Ainsi le back-end est constitué de plusieurs composants softwares dont chacun a une tâche bien spécifique dans le système DAQ:

- ◆ Le Run Control (le RunCtrl) s'occupe de contrôler l'état des sous-systèmes, de gérer l'acquisition des données, par exemple initialise, commence et termine l'acquisition de données.
- ◆ Le DAQ Superviseur qui s'occupe de la configuration et la gestion des paramètres;
- ◆ Information Service (l'IS) qui assure l'échange d'informations entre les différents composants softwares;
- ◆ Le Message Reporting System (le MRS) qui s'occupe de rapporter des messages d'erreurs survenus dans le système;
- ◆ Le Process Manager qui assure la gestion des processus et les ressources nécessaires à leur exécution.

Ces différents composants softwares qui forment le back-end du DAQ exigent une interface graphique utilisateur sophistiquée intégrant tous ces composants afin de les gérer et de les contrôler. En particulier cette interface graphique générale doit être claire et simple à utiliser pour des utilisateurs non experts du système. L'implémentation de cette interface graphique sera basée sur le langage JAVA qui a l'avantage d'être indépendant de plateforme.

Le projet consistera en plusieurs phases depuis la conception jusqu'à l'implémentation :

- 1) Définir les contraintes du problème en terme de qu'est ce qui doit ou peut être afficher par l'utilisateur; quelles sont les actions q'un utilisateur est autorisé à accomplir, comment et quelles informations sont échangées entre l'interface graphique utilisateur et les autres composants softwares qui constituent le système.
- 2) Conception de l'interface définissant comment les informations sont présentées à l'utilisateur tout en gardant l'efficacité et la simplicité du programme.
- 3) Implémentation en JAVA, en utilisant des CASES tools interactifs pour construire des interfaces graphiques et en suite générer le code correspondant. Cela a demandé une investigation des différents tools disponibles sur le marché et notre choix a été porté

sur VISAJ qui offre plus de possibilités interactives nouvelles que beaucoup d'autres tools tels que X-Design.

4) Intégration des programmes JAVA avec les autres composants de Back-end.

Mais avant de commencer à décrire l'interface graphique qui est le but de ce projet, nous donnons une étude du système DAQ/EF pour lequel cette interface graphique utilisateur doit être construite. Nous donnons une étude de l'organisation de l'architecture globale de DAQ/EF, l'architecture de ses sous-systèmes et comment ses sous-systèmes interagissent entre eux dans l'acquisition de données.

En suite nous abordons dans les détails des différents composants softwares qui composent le back-end; leur architecture, leur implémentation et comment ils interagissent entre eux pour accomplir leur mission et nous finirons par l'étude détaillée depuis la conception jusqu'à l'implémentation de l'interface graphique utilisateur qui intègre tous ces composants softwares.

1. Description du projet

Le DAQ du CERN exige une interface graphique utilisateur général qui gère et contrôle le système entier. Le GUI fournit une interface entre l'utilisateur, typiquement les opérations de shift, et tous les composants qui constituent le Back-end software.

Il doit pouvoir échanger des informations entre les différents composants et être capable d'envoyer des commandes pour accomplir des opérations particulières.

1.1 Les exigences et contraintes

Les exigences et contraintes utilisateurs sont résumées comme suit :

1.1.1 Les exigences et contraintes générales

- ◆ Le GUI doit pouvoir afficher la partition courante dans laquelle le système DAQ s'exécute;
- ◆ Le GUI doit être capable de sélectionner des différentes partitions qui existent dans la base de données;
- ◆ Il doit être possible d'exécuter multiples copies concurrentes du GUI;
- ◆ Dans le cas où plusieurs copies s'exécutent, seulement une à la fois doit pouvoir exécuter une action dans le système;
- ◆ Il doit être possible exécuter le GUI sur une machine distante.

1.1.2 Les exigences et contraintes d'affichage

- ◆ Le GUI doit être capable de présenter le statut global du système DAQ et donner un moyen d'afficher le statut des sous-systèmes et composants individuellement;
- ◆ le GUI doit être capable de présenter des informations d'une manière hiérarchique. Il devrait commencer à afficher des informations générales et avoir une option d'afficher des informations plus détaillées au besoin;
- ◆ le GUI doit pouvoir afficher des paramètres généraux d'exécution, des informations générales d'exécutions, et des configurations générales d'exécutions;
- ◆ Le GUI doit être capable d'afficher le statut des informations, le monitoring de ces informations, le type de configuration et éventuellement les messages d'erreurs qui se sont produits pour les sous-systèmes suivant :
 - Front-end DAQ;
 - Read Out Crate;
 - Event Builder;
 - Trigger système (Lv1 et LV2);
 - Event Filter;
 - Le système de controle des détecteurs;
 - Les détecteurs;
 - Les Accélérateurs;
- ◆ Le GUI doit être capable d'afficher le statut de tous les contrôleurs en exécution. Notamment l'information générale doit être le statut du contrôleur Root;
- ◆ Le GUI doit gérer et afficher le statut des processus DAQ qui sont en exécution, Il doit être capable d'afficher des informations détaillées concernant chaque processus en exécution;

- ◆ Le GUI doit être capable de recevoir et d'afficher les messages venant du MRS (Message Reporting System);
- ◆ Le GUI doit être capable de filtrer les messages reçus en fonction des requêtes de l'utilisateur;
- ◆ Le GUI doit être capable de sauvegarder les messages dans un log file;
- ◆ Le GUI doit être capable d'afficher les anciens messages;
- ◆ Le GUI doit être capable d'afficher une information à propos des statuts des processus moniteurs.

1.1.3 Les exigences et contraintes du contrôle

Un contrôle d'identification d'accès doit être effectué avant de pouvoir envoyer des commandes expertes à des sous-systèmes;

- ◆ Le GUI doit être capable de lancer et de stopper le DAQ Superviseur;
- ◆ Le GUI doit être capable d'envoyer des commandes non expertes au Root contrôleur, telles que «start», «stop», «pause» et «resume»;
- ◆ Le GUI doit être capable d'envoyer des commandes d'experts à tous les contrôleurs dans l'arbre du Run Contrôleur;
- ◆ Le GUI doit être capable de créer et de tuer des processus sous des requêtes d'utilisateurs experts seulement;
- ◆ Le GUI doit être capable de lancer des processus gestionnaires;
- ◆ Le GUI doit être capable de lancer et d'afficher des informations d'événements dump;
- ◆ Le GUI doit être capable de lancer l'exécution de book-keeper et d'afficher ses informations.

1.2 Interaction avec les autres composants Back-end

Le diagramme de contexte du GUI apparaît dans la figure 1.1.

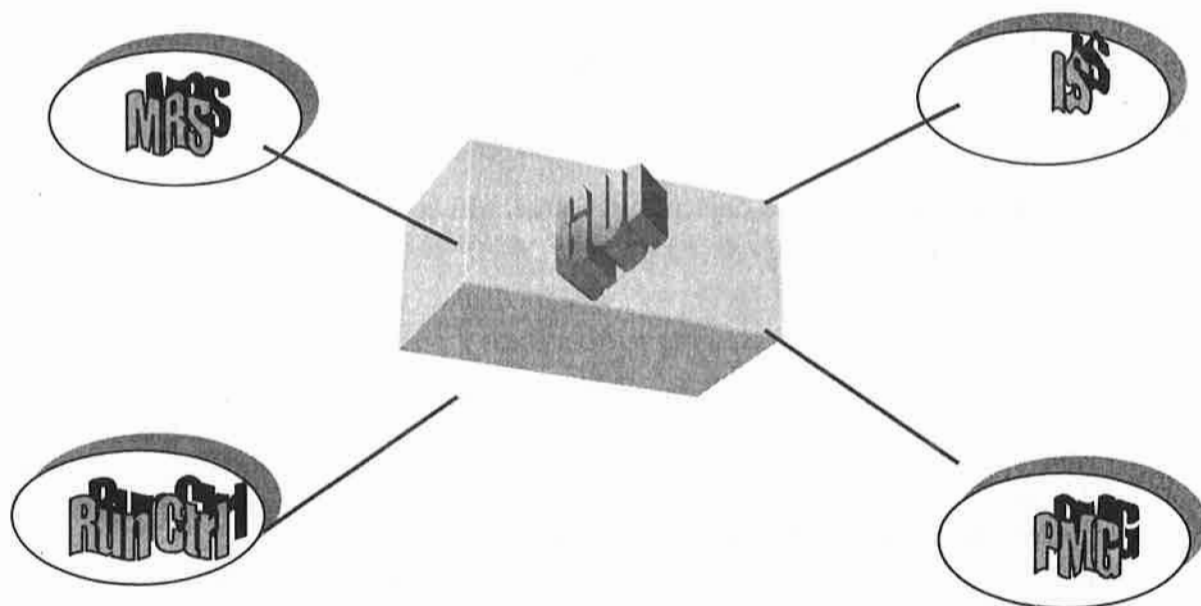


Figure 1.1 Le contexte du GUI.

2. Le prototype-1 du système d'acquisition de données et filtre d'événements (DAQ/EF) d'atlas

2.1 Abstraction

Le Centre Européen de Recherche Nucléaire a décidé d'orienter ses recherches vers la compréhension de certaines questions fondamentales laissées sans réponse dans le contexte de modèles standards et au delà par les moyens d'expérimentation de Hadrons Collider dans le domaine d'énergie multi-TeV. Le Large Hadrons Collider (LHC) est actuellement sous expérimentation et conception au CERN.

Les collisions proton-proton à 14 TeV dans le cadre du centre de la masse ouvrira un nouveau domaine d'énergie donnant accès de façon expérimentale à des nouveaux domaines de physique en particulier la recherche du Higgs Boson, dont l'existence est basée le plus souvent sur la description du mécanisme qui gouverne la génération de masse de particules élémentaires. Pour arriver à une observation sensible des Higgs Boson et d'autres nouveaux phénomènes, le LHC doit tourner à une très haute fréquence qui, combinée avec l'énorme immensité des particules générés dans l'interaction de Hadrons multi-TeV exigent la construction d'énormes détecteurs sophistiqués avec des millions de canaux électroniques qui sont appelés à produire des données gigantesques de l'ordre de 100 Gbyte/s à une haute fréquence de l'ordre de 100 kHz.

Un tel environnement d'expérimentation exige une sévère demande de performance de la part du DAQ, qui doit faire face à des taux d'événements et des flux de données de sortie d'une grande magnitude sans précédent. Un DAQ complexe, caractérisé par multiples niveaux de triggers de collection de données hiérarchisées et une quantité impressionnante d'unités de calculs à temps réel et ayant une interface graphique utilisateur sophistiquée, doit être conçu, avec une architecture innovatrice et qui exige une performance qu'on pourrait espérer d'obtenir avec les technologies d'aujourd'hui.

Les investigations et les études menées ont démontré pour plusieurs aspects du système DAQ des propriétés du temps réel dans les calculs des processeurs du Front-end pour les nécessités du parallélisme des collections des données (Event Building).

Ce projet a été approuvé par ATLAS Collaboration pour la conception et l'implémentation d'un prototype du DAQ/EF basé sur une architecture fonctionnelle décrite dans Atlas Technical Proposal. Le prototype consiste en une architecture puissante du système d'acquisition de données et de filtre d'événements incluant tous les éléments hardware et software de flux de données, ses contrôles, ses moniteurs, aussi bien que tous les éléments d'un système complet en ligne.

Ce mémoire décrira ce projet, ses objectifs, ses structures et décrit en détail l'architecture du système et ses composants.

2.2 Introduction

La version finale du DAQ/EF est prévue pour être utilisée en l'an 2005. Le prototype du DAQ avec son interface graphique utilisateur est prévue d'être fonctionnelle avec le prototype-1 du DAQ au début de l'an 2000. Des spécifications claires de détecteurs, des investigations approfondies de hardware et de software et des études d'intégrations sont toujours nécessaires avant qu'une conception consistante et solide soit mise sur pied.

Le groupe Atlas DAQ a choisi une approche qui consiste à une investigation de prédesign à travers un prototype structuré, supportant l'évaluation des technologies hardware et software et les aspects d'intégrations du système.

Un projet est sur pied à Atlas pour le design et l'implémentation d'un puissant prototype de DAQ/EF basé sur les Triggers/DAQ architectures décrites dans Atlas Technical Proposal et qui supporte l'étude des fonctionnalités de système puissant. A ce stade les performances finales exigées ne sont manifestement pas prises en compte. Pour cette raison il a été nommé Prototype « 1 » (dans la suite nous parlerons du DAQ/EF-1 ou simplement DAQ). Ce projet suit et est basé sur les investigations précédentes du DAQ relatives au problème aux expériences LHC. La figure 2.1 montre l'architecture du DAQ (composée de trois niveaux).

Après une description de l'architecture fonctionnelle de l'Atlas Triggers/DAQ, nous donnons des détails majeurs sur les sous-systèmes et les différents niveaux d'hierarchies du DAQ.

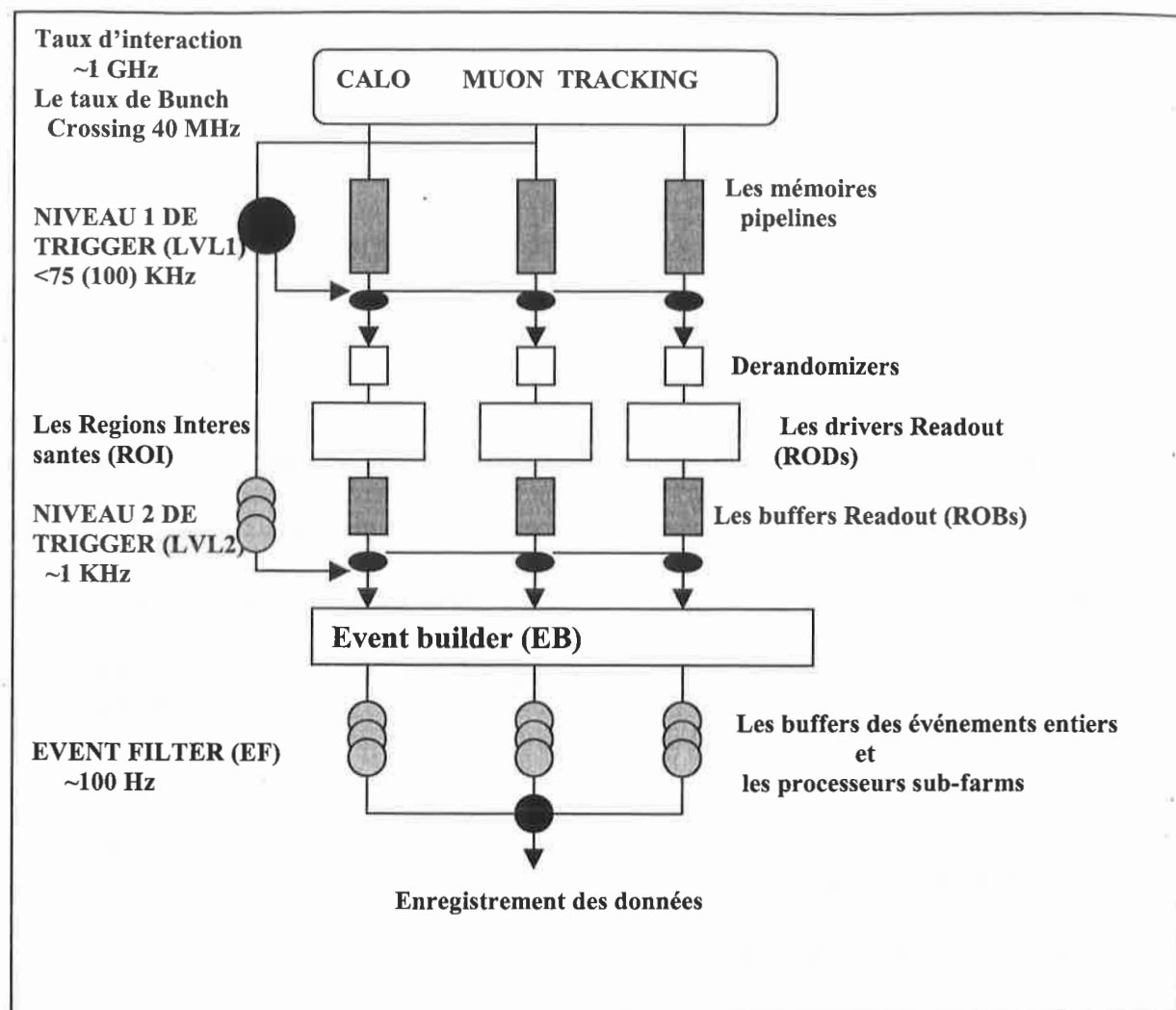


Figure2.1 Architecture de DAQ avec ses trois niveaux.

2.3 Le Système Trigger/DAQ d'Atlas

Le système de sélection de données (ou encore appelé Trigger/DAQ Atlas) doit être capable :

- 1) de réduire l'immense quantité des données produites pendant la collision LHC à une quantité de données gérable pour le stockage permanent;
- 2) de transporter et de distribuer l'énorme quantité de données venant des détecteurs électroniques du Front-end;

La gestion du taux, de la vitesse et la réduction de données est organisée en trois sous-systèmes fonctionnels qui consistent en trois niveaux fonctionnels de Triggers : Le niveau-1 de trigger (le LVL1), le niveau 2 de triggers (le LVL2) et le EF. (Pour plus de détails sur les différents niveaux de triggers voir l'annexe A).

2.4 Le Système Front-end d'Atlas DAQ

La série de flux de données venant des détecteurs vers le ROB est considérable et doit être stockée dans le buffer du LVL1, durant tout le moment de prise de décision de LVL1, une «derandomizer» qui est utilisée pour les données de sortie du système du Front-end, un ReadOut qui héberge les informations d'un nombre de liens du Front-end, une readout link (ROL) connectant le ROD à ROB et le ROB lui-même.

L'interface du Front-end est conçue comme une entrée du ROB tel qu'un seul concept de buffer peut être utilisé à travers l'expérience entière avec un type du ROL et de protocole de transfert de données.

Certaines descriptions de détecteurs spécifiques sont restreintes aux RODs qui doivent fournir une interface aux liens standards. Le LVL2 trigger et le constructeur d'événements seuls accèdent aux données des détecteurs du ROB et ceci clarifie l'interface obtenue.

Basé sur l'exécution du ROL au tour de 1Gbit/s, le nombre total de liens est estimé au tour de 1700.

2.5 Aperçu sur les fonctionnalités de flux de données

La repartition du flux de données est la fonction première du système DAQ. La façon dont les données doivent être transférer à travers le système est déterminée par une architecture décisionnelle. Elle mérite donc une spécification comme un module fonctionnel dans le système DAQ.

L'architecture DAQ Atlas doit supporter les fonctions de flux de données suivantes :

- ◆ A chaque bunch-crossing, les signaux venant de chaque canal électronique de détecteurs sont stockés dans les pipelines des détecteurs spécifiques du Front-end pendant toute la durée du traitement de LVL1 trigger (LVL1 latence).
- ◆ A chaque acceptation de LVL1 trigger, les données digitalisées de tous les pipelines du Front-end sont transférées dans les ROB, où elles sont stockées pendant toute la durée du processus de LVL2 trigger (LVL2 latence).
- ◆ Les données de ROI pour tous les détecteurs exigés par l'algorithme de LVL2 trigger sont transférées des ROB vers le système LVL2 trigger.
- ◆ A chaque acceptation de LVL2, toutes les fractions de données correspondantes aux événements sélectionnés sont rassemblées en des événements complets (Event Building) et transférées vers le système EF.
- ◆ Chaque événement (physique ou calibration) accepté par le processus EF est enregistré dans un stockage permanent.

Dans Atlas, l'architecture du DAQ doit satisfaire à deux catégories de description et d'exigence de flux de données :

- ◆ Les exigences et les contraintes des détecteurs spécifiques du Front-end telles que: la tolérance à la radiation, la digitalisation, et le readout du Front-end, les contraintes géométriques, la puissance de consommation, et le calcul des signaux spécifiques.
- ◆ La description du système trigger/DAQ inclut et exige également le plus grand degré d'uniformité du système, cohérence du format des événements, une organisation adéquate des données pour une sélection efficace et une uniformité de protocoles de transferts de données.

Pour satisfaire à ces exigences, les fonctionnalités élémentaires suivantes ont été introduites dans le flux de données:

1) Les éléments détecteurs spécifiques de flux de données :

- ◆ Les pipelines du Front-end.
- ◆ Les «derandomizers » qui découplent les taux instantanés des données de LVL1 venant des pipelines du Front-end du taux moyen de la capacité de readout LVL1 de l'étape électronique suivant.
- ◆ Le multiplexage des données venant des canaux électroniques du Front-end peut être effectué à plusieurs niveaux le long de la chaîne de flux de données; soit dans des processeurs dédiés, soit comme partie intégrante d'autres éléments.
- ◆ Quelques blocs des tâches du ROD agissent comme un multiplexeur.
- ◆ Le ROD fournit des signaux de détecteurs spécifiques, organise les données comme l'exigent les algorithmes de LVL2 trigger et prépare les données dans le format exigé par le DAQ.
- ◆ Le ROD a été introduit dans l'architecture comme une interface entre les éléments détecteurs spécifiques de flux de données et les éléments standards du DAQ. Son rôle est de convertir l'entrée des détecteurs spécifiques en une sortie standard du trigger/DAQ définie. Il peut également supporter tous les traitements de signaux.

2) Les éléments du système trigger/DAQ (standard pour tous les détecteurs) :

- ◆ Le ROL transfère les fragments d'événements formatés dans le ROD vers le ROB. Le protocole du ROL est spécifié par le DAQ et doit être conforme à tout le système.
- ◆ Le ROB est un élément multi-fonctionnel du flux de données: Il reçoit les fragments d'événements acceptés par LVL1 qui viennent du ROL; Il fait passer les données sélectionnées vers le système LVL2 trigger; Il stocke tous les fragments d'événements pendant la latence du processus de LVL2 trigger; Il transfère tous les fragments des événements acceptés de LVL2 vers le système EF via EB.
- ◆ L'activité principale de LVL2 trigger n'est pas l'acheminement du flux de données. Le système LVL2 consiste en une étape de collection de données et une étape de traite-

ment. La première supporte le transfert de fragments des événements sélectionnés du ROBs vers l'étape de traitement de LVL2, où l'algorithme s'exécute sur les événements partiels de données fournis par la sélection ou rejection d'événements de LVL2. Ces deux fonctions peuvent se répéter multiples fois.

- ◆ Le système EB assemble tous les fragments des événements acceptés par LVL2 du ROBs et les transfère comme des événements entiers (complets) à l'EF. Comme pour LVL2, cette fonction peut se répéter plusieurs fois.
- ◆ Le système EF consiste d'une étape de distribution d'événements et une étape de traitement. La première reçoit des événements complets du système EB et les distribue aux différentes unités de traitements de EF, où des reconstructions complètes sont effectuées pour la sélection finale avant l'enregistrement.

3. Le Sous-système Back-end du système DAQ

3.1 Généralité

Le sous-système Back-end software du DAQ est constitué de différents composants logiciels pour configurer, diriger et contrôler le DAQ. Par contre la gestion, le traitement et le transport des données physiques ne font pas parties de ses responsabilités.

On peut voir le sous-système Back-end comme une colle qui retient ensemble tous les autres sous-systèmes du DAQ. Il permet de rassembler et de relier tous les autres sous-système du DAQ. Le Back-end ne contient pas des éléments qui sont des détecteurs spécifiques comme c'est le cas de certains sous-systèmes.

La figure 3.1 est un diagramme qui montre le contexte de base d'échange d'informations avec les autres sous-systèmes. Ce diagramme est assez général, et ne contient pas toutes les connexions du Back-end.

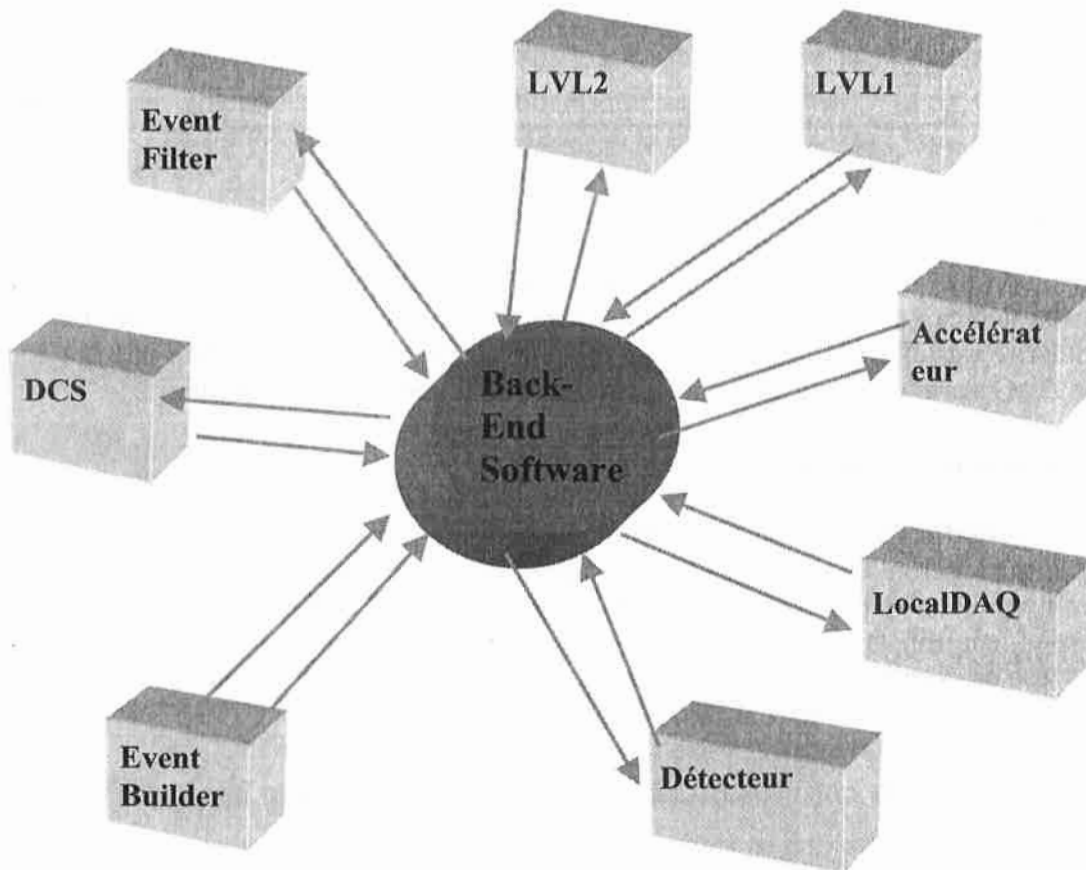


Figure 3.1 : le contexte de base d'échange d'informations avec autres sous-systèmes.

Comme le Back-end est un sous-système du système entier, à ce titre il doit coexister et coopérer avec les autres. En particulier, des interfaces sont nécessaires et exigées entre le Back-end et les autres sous systèmes. En l'occurrence :

Trigger :	Le Back-end reçoit la configuration détaillée des triggers et peut la modifier.
Processeurs farm :	Le software Back-end synchronise ses activités avec les fermes de processeurs lorsqu'il doit référencer ou manipuler les données physiques.
Accélérateurs :	Les détails de faisceaux sont récupérés des accélérateurs et les opérateurs des machines accomplissent un feedback sur les statuts des faisceaux des sites des détecteurs.
Event builder :	Le software Back-end synchronise ces activités avec EB pour la manipulation des données.
Local DAQ :	Pour envoyer et recevoir les informations de contrôles, de configuration et de synchronisation.
DCS	Reçoit les informations de statuts du système de contrôle des détecteurs et envoie des commandes de contrôles.

3.1.1 Modularité et fiabilité de Back-end

Le système prototype DAQ d'Atlas doit être capable de s'exécuter en utilisant tous ou un seul des sous-systèmes qui le composent. Il est conçu d'être complètement modulaire. Ce besoin de modularité est justifié au fait que si un module tombe en panne il ne faut pas que tout le système DAQ s'arrête. Ce qui demande au système un grand degré de modularité.

3.1.2 Gestion des Partitions

Durant les périodes d'utilisation du système, de mis à jour, et de testes, plusieurs groupes de personnes travailleront en parallèle et cela exigera l'utilisation des ressources DAQ variées. Le DAQ contient beaucoup de ressources tant softwares que hardwares qui ne peuvent pas être partagées et leur utilisation doit être contrôlée pour éviter des conflits. Le but de la gestion de partition est de formaliser l'allocation des ressources du DAQ et permettre aux différents groupes de travail et d'utilisateurs de travailler sans s'interférer.

3.2 Les Différents composants qui composent le système back-end

Les spécifications du système Back-end ont été décrites en groupes d'activités ayant des fonctionnalités similaires. Les composants softwares de ces groupes seront décrits plus loin chacun avec ses fonctionnalités et ces limites. Ces composants du Back-end ont aussi des interfaces avec d'autres composants et avec d'autres systèmes externes. Chaque composant offre un ensemble unique de fonctionnalités et sa propre architecture.

Dans l'analyse de ces composants il a été montré que plusieurs domaines réapparaissent dans tous les composants, en l'occurrence : le stockage des données, la communication inter-objets et le GUI.

Ils existent cinq composants qui sont considérés comme le noyau du système Back-end. (Ce noyau des composants constitue l'essentielle des fonctionnalités du système Back-end.) Ces cinq composants sont :

- ◆ Configuration database;
- ◆ Process Manager System;
- ◆ Information service System;
- ◆ Run control System;
- ◆ Message Reporting System.

Comme décrit au début de ce chapitre, le système Back-end est un sous-système du DAQ. Il se comporte comme une colle pour retenir ensemble les autres sous-systèmes du DAQ, ainsi ce rôle de rassembleur des autres sous-systèmes est assuré par l'ensemble des fonctionnalités offertes par les cinq modules du Back-end cités ci-dessus.

Comme on peut donc le constater la gestion, le contrôle et le bon fonctionnement de l'architecture globale du DAQ/EF se reposent sur les cinq modules du Back-end.

D'où l'importance de trouver une interface graphique générale sophistiquée mais simple d'utilisation qui intègre ces cinq composants softwares et offre toutes les fonctionnalités expertes et non expertes exigées de ces cinq composants pour assurer pleinement leurs rôles dans le DAQ.

Rappelons ici que le but de ce projet est de contribuer à la construction de ce Graphique User Interface.

Mais avant de parler du GUI, nous donnons dans les sections qui suivent une description détaillée de chaque composant du Back-end.

3.2.1 Le Composant «Configuration Database»

Un système d'acquisition de données comme celui d'Atlas a besoin d'un grand nombre de paramètres pour décrire son système d'architecture hardware, ses composants softwares, ses modes d'exécutions et ses statuts.

Deux des problèmes majeurs de la conception du système DAQ/EF d'Atlas sont qu'il soit aussi flexible que possible et de paramétrer le software par le contenu de la base de donnée de configuration.

Les informations de configuration sont divisées en plusieurs parties : les composants hardwares, l'architecture software, les paramètres d'exécutions et de sous-détecteurs. Les informations peuvent être groupées suivant la fréquence auxquelles elles changent :

- ◆ Les informations statiques qui ne changent pas pendant une session de traitement de données (il s'agit des informations concernant des composants hardwares et l'architecture software);
- ◆ Les informations périodiquement mises à jour qui ne changent pas pendant une exécution DAQ (il s'agit des paramètres d'exécutions et de sous-détecteur).

Plusieurs autres bases de données sont utilisées par les composants ou sous-systèmes individuellement (exemple: la base de données du sous-système du MRS qui contient les définitions des messages, des informations de calibration de database etc.), mais la description complète de ces paramètres de configuration sort du cadre de ce projet.

3.2.2 Le Composant «Process Manager System» (PMG)

L'objectif du Process Manager System est d'accomplir les contrôles de bases des processus des composants logiciels du DAQ. Il est capable de créer, de lancer, de stopper et de contrôler les statuts de base des processus des composants (c'est à dire running ou exited) sur les stations de travail du DAQ et sur les processeurs du Front-end du DAQ et cela indépendamment des systèmes d'exploitations.

Comme cité ci-dessus le PMG fait partie des cinq composants du Back-end, qui est donc responsable de la création et la terminaison des processus des différentes machines du DAQ. A ce titre il crée des processus locaux, à distance dans des modes synchrones ou asynchrones et chaque processus créé est assigné à un identificateur unique.

Pour lancer un processus, le PMG doit retrouver les informations nécessaires de lancement dans la base de données de configuration. Une notification est générée pour tout changement de statuts des processus en exécution dans le DAQ. Il doit également collaborer avec le gestionnaire de ressources afin de limiter le nombre de copies des processus en exécution au nombre maximal spécifié dans la base de données de configuration, tout en fournissant une interface utilisateur pour gérer et contrôler les processus en exécution.

Plusieurs instances du GUI du PMG peuvent s'exécuter simultanément et doivent fournir un moyen à l'utilisateur de mise à jour des statuts des processus en vie et ces mêmes interfaces doivent être mises à jour dès que le statut d'un processus en exécution change.

Le PMG est un système distribué, composé de différents éléments internes qui ont vraisemblablement été implémentés comme des processus séparés et s'exécute sur des différentes machines. Le PMG est un ensemble de services qui est utilisé par d'autres composants logiciels du DAQ tels que le système Run Control. Il n'a aucune connaissance du DAQ ou des fonctions des processus qu'il gère.

La partie client du PMG fournit une interface entre les programmes externes (par exemple un programme utilisateur) et le module PMG. C'est la seule partie visible du PMG depuis l'extérieur à partir de laquelle une application client peut faire des requêtes au système PMG.

Un processus qui doit être lancé peut être spécifier soit en passant à l'interface client du PMG toutes les informations nécessaires (le fichier exécutable, la machine etc.), soit en passant à l'interface client du PMG des instructions pour retrouver les informations dans la base de données de configuration.

Dans les deux cas de lancement de processus, l'interface client du PMG activera automatiquement une routine callback utilisateur qui créera le processus après avoir reçu de l'interface client du PMG le handler du processus en paramètre et une information flag sur le statut du processus.

L'activation automatique de cette routine est effectuée lorsque les conditions suivantes sont réunies :

- ◆ Lorsqu'un processus en activité meurt, l'interface client appellera la routine en lui donnant le handler du processus et le flag «DEAD»;
- ◆ Lorsqu'un processus dont la création a été demandée en mode asynchrone est maintenant créé et est en activité, alors la routine est aussitôt activée en lui passant le handler du processus et l'information « UP » pour mettre le flag de statuts à la valeur «UP»;
- ◆ Lorsqu'un processus dont la création a été demandée en mode asynchrone n'a pas pu être créé, alors la routine est également activée par l'interface client PMG en lui passant le même handler qui a été retourné lorsque la création précédente a été demandée avec l'information «FAIL».

Lorsqu'une création de processus est demandée de façon asynchrone, le client PMG retourne un handler qui sera associé ultérieurement au processus créé. L'utilisateur ne peut pas l'utiliser avant qu'il n'ait reçu du callback l'information flag «UP» pour signifier que le processus associé à ce handler est bien en vie.

Un nom utilisateur peut être associé au processus créé. Il n'a d'autres fonctions que d'aider un utilisateur humain à identifier plus facilement le processus dans une liste de processus. Ce nom est optionnel.

La partie agent de système PMG s'exécute sur une machine qui a en charge la création effective, la terminaison et la mort des processus qui sont sous le contrôle du PMG. Elle peut être implementée sur la machine locale comme une couche de software qui utilise le système d'exploitation local de la machine pour gérer les processus sur cette machine dont il a en charge. Il doit y avoir un agent PMG sur chaque machine participant au système DAQ. L'agent du PMG est lancé (comme un daemon) au démarrage (boot time) sur chaque machine sur laquelle on doit gérer des processus. D'autre part, il peut être manuellement lancé sur le shell du OS lorsqu'on en a besoin.

L'agent du PMG est également une interface entre le module client du PMG et le module du PMG qui contient les données dynamiques sur les statuts du PMG. Pour ces raisons l'agent a besoin de savoir où cette partie contenant les informations dynamiques du PMG et qu'on appelle le `dynamique_DB` s'exécute et comment il a été implementé.

Le module `dynamique_DB` du PMG est une base de données dynamique qui peut contenir :

- 1) Liste des machines concernées par les services du PMG;
- 2) Liste des agents PMG qui s'exécutent;
- 3) Liste des processus avec la description de leurs données :
 - Handler;
 - le nom utilisateur;
 - la machine;
 - le PID local;
 - le programme du fichier exécutable.

3.2.3 Le Composant «Information Service» (IS)

L'Information Service appelé encore l'IS permet l'échange d'informations entre les différents composants logiciels du DAQ/EF. Les informations venant des sources peuvent être groupées en des catégories d'informations et accédées à travers l'IS par les applications qui en font les requêtes de façon asynchrone.

Le prototype DAQ sera un système distribué d'acquisition de données. Un tel système a besoin d'un sous-système de service d'information qui prend en charge tous les aspects relatifs aux transferts d'informations entre les différents composants de sorte que la communication entre les composants du système soit complètement transparente.

L'IS peut être vu comme un moyen de rendre disponible le contenu d'une information qui lui est fournie normalement par un composant ou un sous-système (source) à d'autres composants et applications auxiliaires (receveur). Donc l'IS est une passerelle entre les composants sources de l'information et les composants receveurs avec l'avantage que le source n'est pas obligé de livrer lui-même l'information au destinataire. Cela a pour effet d'éviter la dégradation de performance des activités d'acquisition de données, car le source ne perd pas son temps à s'occuper du transfert de données et lui permet ainsi de s'occuper d'autres choses pendant que les données sont livrées par l'IS.

Une fois que l'information est publiée chez l'IS par le source, il peut à tout moment mettre à jour l'information et les receveurs peuvent récupérer la plus récente copie.

Du point de vue de transfert d'informations, tout composant dans le DAQ peut être considéré comme un source ou un receveur. Les sources d'informations peuvent être les modules du Run Control, d'autres composants du Back-end, des modules d'autres composants (Front-end, EB, EF), etc...

Les receveurs peuvent être également des modules du Run Control, des composants d'interfaces utilisateurs etc...

Le transfert d'informations peut être fait dans plusieurs modes et livré de la façon :

- ◆ à des intervalles de temps réguliers;
- ◆ lorsque les informations changent;
- ◆ à la demande d'un receveur.

Les sources décideront (dépendant du contenu de l'information) comment livrer l'information (à des intervalles de temps régulier ou lorsque l'information change).

Les sources publient les informations et les destinataires accèdent aux informations individuelles ou logiquement groupées. Pour recevoir des informations les applications destinataires doivent préalablement faire une souscription auprès du IS pour un type d'informations. Ainsi à chaque publication ou changement de l'information, les receveurs qui s'y sont souscrits recevront chacun une copie de la dernière version de l'information.

Un fichier log sur la transaction des informations est aussi produit. Lorsqu'une application fait une souscription pour des informations qui n'existent pas encore, elle recevra ces informations dès que ces informations sont publiées par le source.

3.2.3.1 Relation entre le MRS et l'IS

Lorsque nous aborderons le composant MRS il apparaîtra qu'ils existent quelques chevauchements entre le MRS et l'IS sur des services proposés par le MRS. Plus les fonctionnalités sont similaires, plus grand est la différence dans le style et l'usage de l'information distribuée

par les deux composants. En gros le MRS est utilisé pour envoyer les notifications (par exemple l'arrêt de l'exécution) alors que l'IS fournit l'accès distribué aux valeurs de données. Une autre différence entre le MRS et l'IS concerne la synchronisation entre les sources et les destinataires des informations. Dans le MRS, un receveur doit faire la requête à une information avant qu'elle ne soit produite par le source et ne l'accédera pas tant qu'elle n'aie été notifiée. Dans l'IS la synchronisation entre source et destinataire est beaucoup plus relaxe puisque l'IS garde une copie de la dernière valeur de l'information qui a été produite et qui peut être accédée à tout moment.

3.2.3.2 Structure de l'Information Service

La figure 3.2 ci-dessous présente le diagramme de contexte de l'Information Service. L'IS transfère l'information du source au receveur. Le source publie et envoie l'information à l'IS et les receveurs se souscrivent pour cette information et la reçoivent.



Figure 3.2 : de contexte de diagramme pour l'Information Service.

Les modules principaux de l'IS sont une interface d'application source et une interface d'application receveur, les serveurs d'informations, et le serveur de noms.

Le but des interfaces d'applications est de cacher tous les aspects de communications aux sources et aux receveurs. L'IS a une configuration multi-serveur, dans laquelle des serveurs sont dédiés à des différents domaines du système d'acquisition de données. Pour être accessible l'information est enregistrée dans un serveur de noms.

L'interface de l'IS pour les applications sources permet aux sources de publier les informations, donc de les mettre dans l'état accessible ou les rendre inaccessible et de les mettre à jour.

L'Interface de l'IS pour les receveurs (applications) permet de trouver au niveau du serveur de noms, le serveur d'information sur lequel l'information concernée existe et contacte ce serveur par un message de souscription ou par une requête d'obtention de la plus récente valeur de l'information.

Si la requête d'information est une valeur simple alors la valeur de l'information est livrée.

Si la souscription est un autre mode de livraison d'informations, alors la requête est ajoutée à la liste de souscription à ce type d'informations et lorsque l'information change, un broadcast est fait à tous les receveurs de la liste de souscription de cette catégorie d'informations.

Le serveur d'information peut également fournir des informations sur les services des opérations des requêtes disponibles.

3.2.3.3 Configuration Multi-serveur

Une solution possible pour le problème de communication relatif à l'implémentation d'un système d'Information Service dans une configuration distribuée peut être l'usage du système d'Inter-Language Unification (ILU).

La figure 3.3 présente une architecture multi-serveur utilisant le service de nom de ILU (ILU Naming Service).

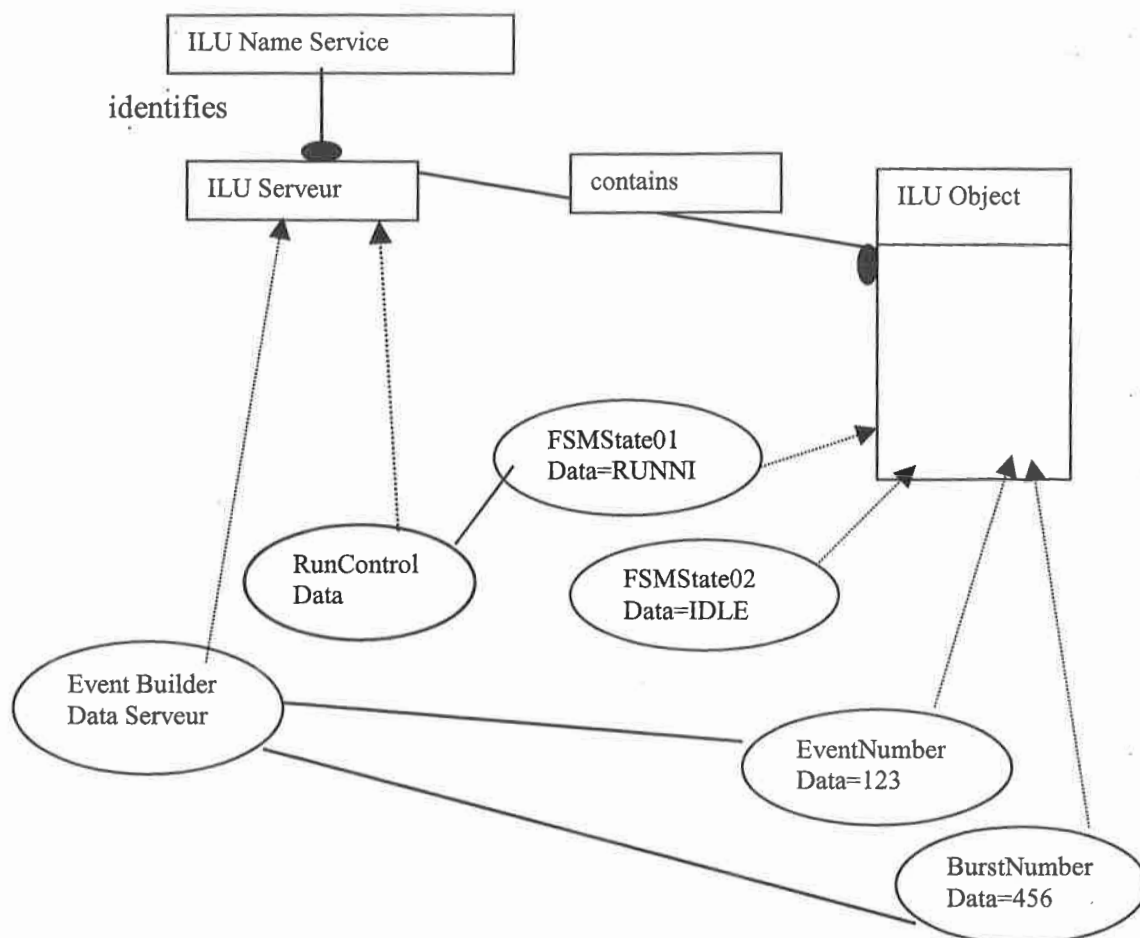


Figure 3.3 Architecture multi-serveur utilisant le service de nom de ILU (ILU Naming Service).

Cet exemple montre que l'IS peut être implémenté en utilisant le concept naturel du système ILU. L'architecture d'IS est très semblable à l'architecture ILU lui-même. Le Service de nom d'ILU peut être utilisé pour accéder aux serveurs qui sont sensés d'héberger les informations. L'information peut être stocker dans des objets qui sont naturellement des objets ILU avec des méthodes implémentées qui peuvent être appelées à distance.

L'algorithme qui permet de déterminer comment configurer plusieurs serveurs et comment les informations sont stockées sur chaque serveur ne sont pas défini ici et l'implémentation du IS sort du cadre d'étude de ce projet.

3.2.3.4 Partitionnement DAQ

L'usage de multiples serveurs permet à l'IS de supporter le partitionnement des informations lorsque plusieurs configurations concurrentes du DAQ s'exécutent.

3.2.3.5 Exemple de fonctionnement d'IS

Pour fixer les idées sur le fonctionnement de l'IS, voici un exemple d'application de l'IS pour une configuration avec deux modules du Run Control Rcl_module_1, Rcl_module2, et un User interface :

Les modules de Run Control sont à la fois des sources et des receveurs en même temps. L'interface utilisateur est seulement un receveur.

Un scénario Run Control peut être par exemple l'envoi de commandes, dans quel cas un utilisateur peut envoyer une commande à travers l'interface utilisateur au contrôleur global. Pour ce scénario il est supposé que les modules du Run Control sont seulement des sources d'informations. (Lorsqu'un contrôleur envoie une commande à ses contrôleurs fils, les contrôleurs fils informent directement le père des nouveaux statuts sans utiliser la base de données dynamique du IS (On verra plus tard que Run Control a une structure arborescente).

Rcl_module1 et Rcl_module_2 publient en tant que sources leurs informations (état). L'interface utilisateur fait une souscription pour les informations venant de ces deux sources. Depuis le User interface la commande est envoyée au module Rcl_module_1 qui en suite l'envoie au module Rcl_module_2. Lorsque la commande envoyée par module1 est exécutée, Rcl_module_2 change de statuts et envoie le nouveau statuts à la base de données dynamique de l'IS, qui retransmet l'information à User interface. Alors le module Rcl_module_1 change d'état tout seul lui même et envoie cette information à la database dynamique d'IS, qui retransmet également cette information à l'interface utilisateur.

3.2.3.6 Les Sources d'informations et les types d'informations

Dans cette section, il est proposé un bref aperçu des informations et les types d'informations que transporte le système IS dans DAQ prototype-1. Le but est d'essayer d'énumérer les différents types principaux. A noter, que d'ici à la fin du développement du projet DAQ ces types peuvent être étendus en fonction des besoins, et que certains nouveaux groupes de sources d'informations peuvent être ajoutés si nécessaire.

Front-end

Les informations en provenance du sous-système Front-end sont relatives au nombre d'événements, au taux d'événements, à la taille d'événements, aux activités du buffer et son occupation, aux statuts des tâches du Front-end, et aux informations relatives aux sous-détecteurs.

Event Builder

Les informations de l'EB peuvent être relatives au taux d'événements, tailles des événements, les activités, états des tâches de l'EB.

Event Filter

Les informations du sous-système EF peuvent être relatives à la taille du farm, l'occupation de farm, taux événements, taux des triggers et l'espace disponible sur le dispositif d'enregistrement.

Back-end

Run Control

Les informations suivantes peuvent provenir du Run Control :

- ◆ Nom de la base de données de configuration sélectionnée;
- ◆ Les statuts des RC contrôleurs;

- ◆ Les informations de processus de RC contrôleurs .

Process Manager

Les informations qui peuvent venir du Process Manager sont les suivantes :

- ◆ La liste des machines concernées par MPG_service;
- ◆ La liste des agents du Process Manager MPG_Agents qui s'exécutent;
- ◆ Liste de processus MPG avec leur description de données :
 - handler (peut être un nom d'utilisateur défini);
 - machine;
 - PID local;
 - temps du commencement;
 - Le fichier du programme exécutable.

Message Reporting System

Les informations suivantes pourraient être fournies par le MRS :

- ◆ Le plus haut degré de sévérité du message transmit par MRS dans un certain intervalle de temps;
- ◆ Le nombre d'erreur d'un certain niveau de sévérité dans un intervalle de temps;
- ◆ Le nombre de sources connectés;
- ◆ Le nombre de modules qui reçoivent les messages;
- ◆ Le nombre de serveurs privés de MRS (doit être une option pour les experts).

Configuration database

Les informations de database sont principalement des paramètres qui ne changent pas à priori durant une simple exécution du DAQ.

3.2.3.7 Le Format d'Informations

De la liste des informations citées ci-dessus, on déduit qu'ils existent plusieurs types d'informations, mais des types beaucoup plus généraux pourront être définis plus tard.

Une pièce d'information sera caractérisée par les attributs suivants :

- ◆ Nom de domaine (unique);
- ◆ Nom de l'information (unique avec un domaine);
- ◆ Type de valeur (ou plage de valeurs);
- ◆ Le temps de la dernière mise à jour.

Un receveur doit se souscrire à une information spécifique ou à une information d'un domaine spécifique. L'IS ne supporte pas toute la définition complète telle qu'elle est faite dans la base de données de configuration, mais fournit un moyen de réorganiser et de restructurer ces informations.

3.2.4 Le Composant «Run Control » (RC)

Le système RC est l'un des composants softwares du Back-end. Il a pour responsabilité de contrôler l'acquisition des données en coordonnant les opérations des sous-systèmes du DAQ, et des systèmes externes au DAQ. Il peut envoyer des commandes aux composants du DAQ, demander et recevoir les informations de statuts de ces composants et coopérer au diagnostic des packages.

L'organisation du sous-système sera décrit dans une base de données de configuration qui est différente et indépendante de la base données de configuration du DAQ. Une fois les activités d'acquisition de données sont en exécution les activités du sous-système ne doivent pas causer

la perte des données en traitement. Le RC doit également permettre la reconfiguration du DAQ en faisant les opérations de shift, il doit assurer l'exclusivité d'usage des partitions avant de permettre de lancer les activités d'acquisitions de données.

Le sous-système RC est un système distribué reflétant la structure du système global du DAQ lui-même. Pour cela, il est composé des éléments séparés qui sont implementés comme les processus s'exécutant sur des différentes machines.

Il a été choisi d'utiliser le système ILU qui est conforme aux normes de communication CORBA. Il sera seulement fait la différence entre les appels synchrone et asynchrone des méthodes. Dans une méthode synchrone, le thread appelant est bloqué jusqu'à ce que la méthode aie fini son exécution et aie retourné ses résultats, alors que dans une méthode asynchrone, le thread appelant continue son exécution après avoir posté la requête, sans attendre de réponse.

Le RC implemente l'approche maître/esclave entre l'opérateur et le DAQ où l'opérateur est le maître. Il utilise les informations cachées et les commandes de factorisation afin de simplifier le contrôle d'un nombre d'éléments DAQ séparés. Dans ce but, il a été défini un ensemble d'états généraux et commandes ayant des sémantiques communes qui peuvent être mappés localement dans les actions spécifiques des composants DAQ. Il supporte le partitionnement du système DAQ durant la mise à jour et est capable de s'exécuter avec un ensemble de variables des composants de contrôle. Cela implique qu'il est capable de lire la configuration des données du DAQ au moment de démarrage du système et de se configurer lui-même en fonction de ses besoins.

Le RC devrait fournir un moyen aux utilisateurs de gérer les sous-systèmes du DAQ indépendamment du système global, de permettre de réagir au changement local et global de statuts du DAQ et de prendre les actions relatives à ce changement de statuts.

3.2.4.1 Structure de Run Control

L'architecture du RC peut être vue comme une hiérarchie des entités de contrôle appelées «contrôleurs» chacun avec une responsabilité bien définie dans le composant. Le statut d'un contrôleur est la vue simplifiée de l'actuelle condition de travail du composant ou du DAQ sous ses responsabilités.

Chaque contrôleur peut recevoir des commandes venant d'autres composants ou d'autres applications. Ces commandes reçues peuvent causer l'exécution des actions qui potentiellement changeront le statut actuel du composant. Un contrôleur peut également réagir aux événements locaux qui se produisent dans des composants DAQ sous ses responsabilités. Typiquement ses réactions peuvent être d'exécuter quelques actions qui potentiellement changeront ses états visibles, La figure 3.4 montre l'interactions entre un contrôleur et le composant qu'il contrôle.

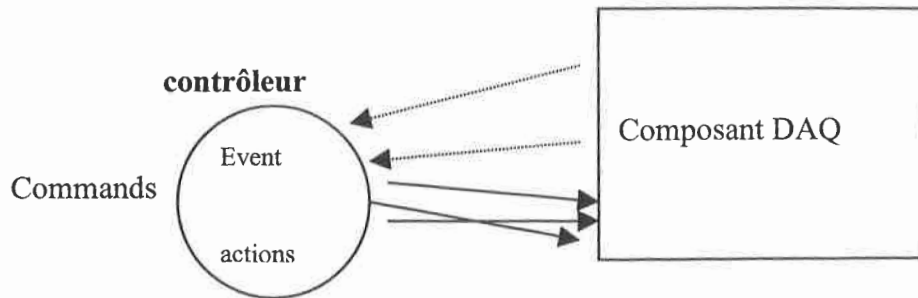


Figure 3.4 : L'interactions entre un contrôleur et le composant qu'il contrôle :

Les contrôleurs sont organisés en une structure d'hierarchie arborescente qui reflète l'organisation générale du système DAQ. Chaque contrôleur dans l'arbre a un parent (ou supérieur) contrôleur et un nombre de fils (subordonnés) contrôleurs. A la racine de l'arbre, un contrôleur ancêtre (le Root) de tous les autres représente le statut global de tout le système en ligne. En dessous du Root, ils existent des ensembles de contrôleurs, un pour chaque sous-système du DAQ et de détecteurs physiques. En dessous de chaque sous-système de contrôleurs il y'a en outre des contrôleurs de composants qui sont responsable des composants individuels.

Les contrôleurs de d'arbre se transmettent les messages entre eux pour s'échanger des commandes et des informations de statuts. En général, les commandes lancées par un opérateur humain sont envoyées au contrôleur général appelé (Root) qui les fait suivre vers ses sous-systèmes de contrôleurs qui en retour les fait également suivre vers les contrôleurs de composants et ainsi de suite. Les résultats des commandes remontent en arrière plan des feuilles vers le Root de sorte que l'opérateur humain est mis au courant de tous les changements qui interviennent dans les statuts du système. Certains nœuds dans l'arbre peuvent accomplir quelques actions sur les commandes ou résultats des commandes qu'ils reçoivent.

Nous assumons, à partir de ces observations de la structure du système, que le niveau de profondeur de l'arbre des contrôleurs peut devenir assez important, mais les concepteurs du composant RC n'ont pas mis de limitation sur le nombre de niveaux de l'arbre ou sur le nombre de contrôleurs à chaque niveau.

3.2.4.2 Le CHSM générique de contrôleur DAQ

Il est envisagé que le comportement dynamique de chaque contrôleur DAQ dans l'arbre de Run Control sera modélisé par quelques diagrammes des états généraux. Les diagrammes des états sont un raffinement des machines à états finis offrant quelques facilités de modélisation incluant la concurrence des états. CHSM (Concurrent Hierarchical finite-State Machine) est un langage et un package de software utilisé pour définir et d'implémenter une telle machine. Il est utilisé ici pour concevoir et implémenter le composant Run Control. Il est prévu que les développeurs des différents contrôleurs de différents groupes, programment le comportement de leur contrôleur particulier en ajoutant des codes pour implémenter le comportement attendu dans le cadre général de ce framework. La figure 3.5 représente le diagramme CHSM.

Le CHSM générique des contrôleurs du DAQ a été dessiné en utilisant l'outil StP CASE . Le CHSM complet consiste à un nombre d'états concurrents. Chaque nœud porte le nom de l'état comme étiquette. Une transition entre deux états est indiquée par une flèche et étiquetée en utilisant la notation standard de OMT de la manière suivante : `event_name[guard_condition] / action`. `Event_name` est le nom de l'événement qui fera que la transition a lieu. Le `guard_condition` est une expression booléenne qui est évaluée avant que la transition ne se produise. Si elle est évaluée à «true», alors la transition aura lieu, si elle est évaluée à «false» alors la transition n'aura pas lieu. L'action est une fonction ou un bloc de code qui sera exécuté lorsque la transition se produit. Les actions à accomplir en entrant ou en sortant dans un état sont mises dans la table d'état StP. La notation utilisée pour spécifier la condition de garde et les actions est celle du système de langage de CHSM.

Il est possible de programmer son comportement tel que chaque contrôleur du DAQ dans l'arbre de run control puisse accomplir les fonctions qui lui sont attendues. Dans le système de langage de CHSM les états, les clusters, les ensembles et le parent CHSM lui-même sont des instantiations des classes C++ `CHSM_state`, `CHSM_cluster`, `CHSM_set` et `CHSM_CHSH` respectivement. Ces classes sont définies au moment de l'exécution de la librairie. Plutôt que d'instantier directement les classes, le Run Control CHSM peut être programmé en les utilisant comme des classes de base à partir desquelles on peut dériver des nouvelles classes avec des fonctionnalités utilisant l'héritage. Une classe par défaut a été définie pour chaque état, cluster et ensemble dans les activités des clusters et les classes héritières des classes `CHSM_state`, `CHSM_cluster`, et `CHSM_set` respectivement. Les classes par défaut implémentent un nombre de fonctions membres supplémentaires, de plus fournissent des fonctionnalités par défaut. Par exemple les conditions de garde et les actions sur les événements sortants d'un état particulier deviennent des fonctions membres pour cet état. Les actions «entrée» et «sortie» définies dans le tableau d'état correspondant aussi deviennent des fonctions membres.

Une classe des utilisateurs est définie pour chaque état, cluster, ensemble et hérite de la classe de défaut correspondante. Dans la classe utilisateur, les développeurs du Run Control peuvent redéfinir ces fonctions par défaut pour lesquelles ils veulent donner des actions spécifiques. S'il n'y a pas d'action spécifique à accomplir alors la redéfinition n'est pas nécessaire et la version par défaut de ces fonctions sera appelée.

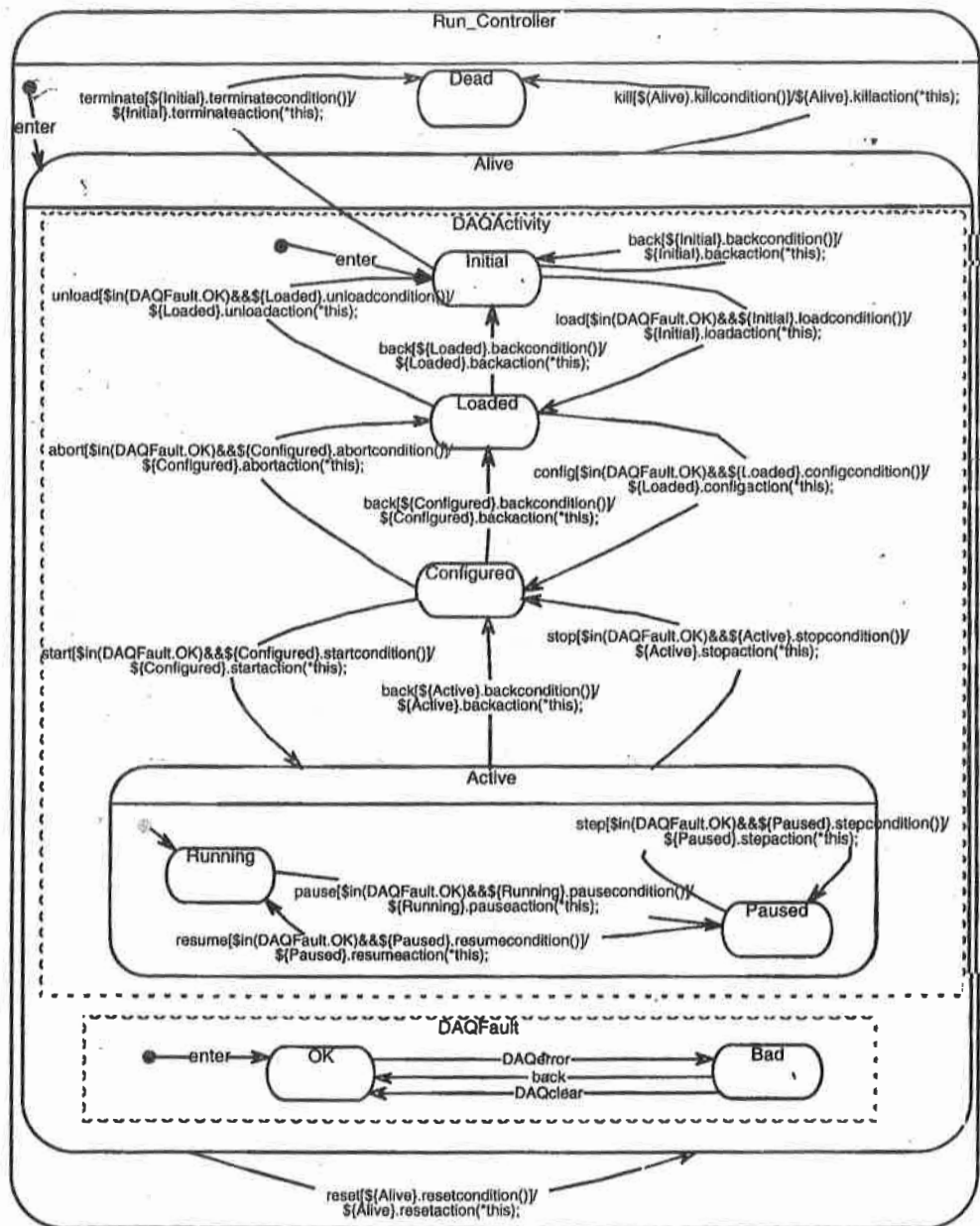


Figure 3.5 Le diagramme CHSM.

3.2.4.3 Récupération d'erreurs

Le CHSM des contrôleurs du DAQ a été désigné pour faire face à trois différents niveaux d'erreurs qui peuvent se produire pendant l'exécution du système :

- ♦ Si une erreur se produit pendant la transition entre deux états, un mécanisme est prévu de revenir en arrière dans le dernier état sans erreur sans prendre en compte les actions qui ont causées cette erreur.
- ♦ Si l'erreur est plus sérieuse et ne peut pas être clarifiée par le mécanisme ci-dessus le système CHSM en entier doit être réinitialisé durant laquelle toutes les ressources allouées sont réinitialisées et le CHSM sauvegarde en arrière plan son état initial.
- ♦ Si une erreur fatale non récupérable se produit quelque part dans le système global de Run Control, le système en entier doit être redémarré (shutdown). Comme l'intégrité du système ne peut pas être garantie dans de telle situation, aucun contrôleur individuel ne peut compter sur des communications ou interaction externes durant le shutdown.

La référence au diagramme ci-dessus, considère l'état central comme un super-état appelé «Alive», (le super-état en vie). Cet état est composé de deux états concurrents : le DAQActivity et le DAQFault. Le fait que ces états soient concurrents est indiqué par des lignes pointillées. En terme du langage du CHSM «Alive» est un ensemble. Lorsque l'état «Alive» est actif alors les deux états le DAQActivity et le DAQFault sont aussi actifs. L'objectif de ces états concurrents est de permettre la récupération d'erreurs lorsqu'un problème se présente sur une transition. Le DAQActivity et le DAQFault sont des super-états, (ou clusters dans le langage de CHSM). L'état enfant actif dans le DAQActivity reflète le statut local du Run Control. L'état enfant actif dans le DAQFault indique s'il y'a eu ou non une faute qui s'est produite lorsque le DAQActivity cluster accomplissait sa dernière transition.

Beaucoup d'erreurs sérieuses sont traitées dans les niveaux supérieurs en remontant dans le diagramme. Lorsque le mécanisme exposé ci-dessus échoue la récupération d'erreur, alors l'événement «reset» est lancé. Cela provoque la sortie de l'ensemble d'états «Alive» indépendamment de l'état courant actif dans les clusters DAQActivity ou DAQFault. Une fonction «exit» est appelée pour l'ensemble d'états «Alive», suivie par une action de transition et une action «enter» lorsque l'ensemble Alive est refranchit de nouveau. Toutes ces actions peuvent être définies par l'utilisateur et exécutées quelles que soient les actions nécessaires pour remettre le contrôleur à son état initial. Lorsqu'on entre de nouveau dans l'état «Alive», il sera dans les états de défaut des clusters DAQActivity et DAQFault (c'est à dire dans l'état Initial pour le DAQActivity et No pour le DAQFault).

Finalement, lorsqu'une erreur fatale se produit dans le système Run Control, l'événement kill peut être exécuté, qui fera que le CHSM changera d'état de «Alive» à «Dead». En sortant de «Alive» la fonction «exit» correspondante sera appelée. Plus loin une action de transition sera appelée lorsque la transition à tuer sera complète. L'action définie par l'utilisateur sera accomplie pour toutes les tâches nécessaires pour faire une sortie (shutdown) du système aussi proprement que possible, mais sans communiquer avec d'autres modules du fait que le système d'intégrité ne peut pas être garanti dans de telle situation.

Le comportement attendu d'un contrôleur pour chaque état défini dans CHMS est résumé comme suit :

« Initial » : Lorsque le contrôleur est dans cet état, cela veut dire que le composant est visible et peut communiquer avec le RC.

« Loaded » : Le contrôleur a lu la configuration nécessaire pour son initialisation.

« Configured » : Lorsque le contrôleur est dans cet état, cela veut dire que le contrôleur a exécuté toutes les phases d'initialisation et est prêt à opérer.

« Running » : Le contrôleur est en action, en exécution et en train d'accomplir toutes ces fonctions d'acquisition de données dans la configuration DAQ sélectionnée.

« Paused » : Le contrôleur est en mode «pause» après un moment d'exécution, il est capable de revenir à l'état d'exécution («Running») et de retrouver tout le contexte de son état «Running» comme s'il n'avait jamais fait de pause.

« Yes » : Le contrôleur a échoué l'exécution de sa dernière commande ou une erreur s'est produit.

« No » : Le contrôleur est revenu à son précédent état (après être dans un état d'erreur).

« Dead » : Le processus ou le thread qui exécute le contrôleur a été tué.

Le résumé des transitions

Toutes les transitions sont définies pour tous les contrôleurs. Si un contrôleur particulier n'a pas besoin d'accomplir certaines actions durant une transition, alors il finit la transition immédiatement avec succès. De telles maquettes d'actions seront des comportements par défaut. Des fonctions sont associées à des actions de transition par le développeur du contrôleur. De telles fonctions sont exécutées automatiquement lorsque la transition est tirée. Elles accomplissent toutes les actions nécessaires pour changer l'état interne du module pour être en accord avec la sémantique demandée pour les nouveaux états. Le développeur peut optionnellement spécifier les actions d'entrée et de sortie pour chaque état. Les actions d'entrée et de sortie sont appelées lorsque la transition qui a causé le changement d'état est tirée.

Les types de comportement attendu de chaque transition :

- ◆ «load»: Le module initialise certains packages de software dont il a besoin pour opérer. (par exemple accès à la base de données, réseau etc.);
- ◆ «unload» : Le module désengage lui-même certains softwares, des packages qu'il a utilisés durant ses opérations (database et réseau);
- ◆ «config» : Le module lit les paramètres nécessaires (c'est-à-dire de database ou de commande line), lance certains services nécessaires (des blocs de processus ou thread), se connecte à d'autres modules (c'est à dire l'établissement des connections réseau) et les requêtes de certaines ressources nécessaires;
- ◆ «Abort»: Le module désalloue certains ressources, arrête certains threads ou processus créés et se déconnecte d'autres modules;
- ◆ «Start»: Le module lit certains paramètres dynamiques et commence les opérations (l'acquisition de données);

- ◆ «Stop»: Le module arrête les opérations et devient prêt mais inactif;
- ◆ «Pause»: Le module arrête les opérations temporairement et devient prêt mais inactif;
- ◆ «Résumé»: Le module résume les opérations;
- ◆ «Step»: Le module résume l'opération d'un événement ou unité de travail et se met dans pause de nouveau;
- ◆ «DAQerror»: Une erreur s'est produite pendant une activité du module qu'il ne peut pas lui-même corriger et qui empêche sa continuation;
- ◆ «Back»: Retrace les actions effectuées pour revenir au précédent état valide;
- ◆ «Aqclear»: L'état DAQFault a été corrigé sans avoir accompli un retour en arrière;
- ◆ «Reset»: Toutes les ressources internes sont reinitialisés ou libérés et le module fait un retour en arrière jusqu'à l'état initial;
- ◆ «terminate»: Provoque le «shutdown» d'un module et le processus de contrôleur à se terminer durant une séquence normale de «shut down»;
- ◆ «kill»: Le module essaie de stopper aussi proprement que possible certaines activités associées à son état courant et exécute ensuite la fonction «exit» (c'est-à-dire terminaison de processus). Le module ne peut pas relier, compter ou dépendre de certaines communications ou interactions externes dans cette transition, parce que le système global RC est supposé corrompu quelque part.

Le DAQ Superviseur

Le DAQ superviseur est responsable de la création de tous les processus logiciels (incluant tous les contrôleurs et les composants Run Control) durant le startup du DAQ suivant la configuration définie dans la base de données. Il utilise le PMG pour créer des processus sur des processeurs du système DAQ. Il est aussi responsable d'assurer normalement le shut down du système à la fin de l'acquisition de données. Une machine séparée à état fini est aussi utilisée pour modéliser le comportement du DAQ superviseur.

Les actions exécutées

Au moment de démarrage (boot) :

En entrée :

- 1) Si cela n'est pas déjà fait, lire la configuration de la base de données;
- 2) Contrôler la disponibilité des services offerts par le PMG, l'IS et le MRS;
- 3) Si cela n'est pas déjà fait, enregistrer ses services dans un serveur ILU de noms et les rendre disponibles;
- 4) Créer le Dynamic_DB et publier l'état du DAQ Superviseur (initialisé au moment du boot);
- 5) Lancer à partir du PMG, suivant les exigences, tous les processus nécessaires pour le système Run Control pour travailler. Cette action peut être avortée dans les cas où un des événements «RCProcessDead», «rcwol» ou «timeout» est génér.

En sortie :

- 1) Stopper le timeout de la création des processus, si ce n'est pas déjà fini.

Pendant l'exécution

En entrée :

- 1) Contrôler l'intégrité du système de configuration des processus. Si la configuration n'est pas correcte, alors générer l'événement «RCProcessDead» pour aller directement dans l'état «Fault»;
- 2) (Si le contrôle se termine bien) mettre l'état du système RC dans Dynamic_DB à la valeur «UP».

En état Fault

En entrée :

- 1) Mettre l'état du système RC dans Dynamic_DB à «Fault».

Au moment de shut down

En entrée :

- 1) Mettre l'état du système RC dans Dynamic_DB à «ShutDown»;
- 2) Envoyer des commandes synchronisées pour stopper tous les contrôleurs. Ces commandes seront traduites comme l'événement «terminate» des contrôleurs génériques.

En exit :

- 1) Stopper le timeout si ce n'est déjà pas fini.

Lorsque dans cet état, le DAQ Superviseur est avisé de la mort de tous les processus, il attend alors la disparition du dernier pour générer l'événement «AllcontrôleursDead».

Dans l'état «FatalFault»

En entrée :

- 1) Mettre l'état du système RC dans Dynamic_DB à «FatalFault»;
- 2) Envoyer la commande «kill» asynchrone à tous les contrôleurs.

Lorsque dans cet état, le DAQ Superviseur est avisé de la mort de tous les processus, il attend alors la disparition du dernier pour générer l'événement «AllcontrôleursDead».

Dans l'état Idle

Pour entrée dans cet état il suffit seulement de mettre l'état du système RC dans Dynamic_DB à la valeur «Idle».

3.2.4.4 Les Communications du DAQ Superviseur

Les événements auxquels le DAQ Superviseur réagit sont générés par l'appel venant du PMG (les fonctions callback, voir PMG) ou par notification venant de Dynamic_DB. Cela ne pose pas des problèmes d'adressage lorsque tous les deux PMG ou Dynamic_DB connaissent l'adresse du DAQ Superviseur d'une précédente méthode d'appel pour la création de processus ou la description des valeurs des états. Ils n'ont pas besoin d'un nom prédéfini pour localiser le DAQ Superviseur.

Il y'a différents problèmes pour les deux commandes qui peuvent être reçues de l'interface utilisateur pour démarrer ou stopper le système, (les événements «BootDAQ» et «Shutdown DAQ»). Ces deux commandes sont envoyées par une interface utilisateur qui est une façon de s'adresser au DAQ Superviseur par nom avec un nom prédéfini.

3.2.4.5 Les éléments du composant Run Control

Basé sur la description ci-dessus il a été proposé de fournir les éléments suivants du composant Run Control :

- ♦ Une implémentation d'un squelette qui peut être utilisé dans n'importe quel exemple de situation décrit ci-dessus. Ce squelette opérera suivant la machine à état fini généralisé et devrait être capable de recevoir et d'envoyer des commandes dans le réseau;
- ♦ Les actions accomplies par les contrôleurs auront besoin d'être implémentées par les groupes de développeurs dans des sous-systèmes;
- ♦ Un moyen d'organiser un ensemble de contrôleurs dans une hiérarchie représentant la structure logique de l'expérience. L'hiérarchie sera définie en se basant sur la configuration de la base de données;
- ♦ Une interface graphique utilisateur pour gérer, contrôler et interagir avec l'ensemble des contrôleurs qui forment l'hiérarchie du contrôle;
- ♦ Une simple interface graphique utilisateur qui est seulement capable de communiquer avec un seul contrôleur. Le but visé est un GUI qui permet à un contrôleur être testé de façon isolée;
- ♦ Une interface software ainsi que des programmes auxiliaires (tels qu'un afficheur de statut) qui peuvent récupérer des informations de statuts de chaque contrôleur;
- ♦ Un programme superviseur capable de lancer plusieurs éléments softwares du DAQ suivant la configuration prédéfinie dans la base de données.

3.2.4.6 Interaction entre les contrôleurs

En plus du CHSM utilisé pour modéliser les activités d'acquisitions de données, un autre CHSM est prévu pour modéliser l'interaction entre les différents contrôleurs dans l'hiérarchie du système Run Control.

La notation est la même que celle utilisée dans le CHSM générique du DAQ. Pour la communication entre contrôleurs, nous tentons d'utiliser les services d'ILU dédiés pour envoyer des commandes vers les niveaux inférieurs de l'arbre du Run Control et à l'IS, pour remonter les répliques de l'arbre et pour rendre visible les états du «DAQActivity» à l'extérieur.

Ils existent cinq 5 états courants dans ce CHSM représentant les interactions entre les contrôleurs :

- ♦ L'état «FSMMgr» concurrent qui référence la configuration initiale d'un contrôleur lors du démarrage et qui le mène soigneusement à travers les différentes transitions quand il reçoit les commandes. Cela inclus l'envoi de la commande reçue par ses fils;

- ◆ L'état «membreship» concurrent qui indique si le contrôleur fait partie de l'hierarchie des contrôleurs du Run Control. S'il est dans l'hierarchie (c'est à dire «In») le contrôleur sera contrôlé par ces parents contrôleurs spécifiés dans la base de données de configuration. S'il n'est pas dans l'hierarchie (c'est à dire «Out») il sera ignoré par ses parents. Un enfant peut être inséré dans la liste du «membreship» (à sa position originale) lorsque l'opérateur en donne l'ordre en envoyant la commande d'insertion. Cette procédure peut être très puissante lorsqu'un contrôleur trouve une erreur qui a besoin d'une attention. C'est aussi un moyen pour tous les autres contrôleurs (qui franchissent sans erreur une transition) de ne pas être obligés de revenir sur les traces de leurs états non nécessaires. A noter que pour la référence future, un enregistrement non volatile gardera le statut du «membreship» de chaque Run Control;
- ◆ L'état CommsMgr concurrent indique le style de communication qui est employé pendant l'envoi des commandes aux fils contrôleurs. Par défaut, toute communication est asynchrone d'où CmmMgr est un asynchrone sous-état. Lorsque c'est nécessaire, le développeur d'un contrôleur peut spécifier que la communication synchrone peut être utilisée pour envoyer des commandes aux enfants contrôleurs en appelant une méthode dans le code d'une action définie par l'utilisateur d'une transition;
- ◆ La méthode génère un événement «sensync» qui fait que le CmmMgr bascule dans le sous-état synchrone. L'état FSMMMgr concurrent contrôle l'état CommsMgr durant une transition pour décider quel est le moyen de communication à utiliser. Une fois la transition concurrente est franchie, le CommsMgr revient au mode de communication asynchrone;
- ◆ L'état concurrent RCFault représente le statut de l'hierarchie du Run Control lui-même. Si ce contrôleur a un problème avec certains de ses enfants (par exemple un enfant qui est mort, qui ne répond plus, etc..) alors l'état «RCFault» est mis à la valeur «Bad». Si tous les enfants sont essentiellement en vie et sans faute, alors cet état est à la valeur «OK». Les erreurs qui concernent le fonctionnement réel du composant DAQ contrôler par les contrôleurs sont enregistrées dans l'état «AQFault» du DAQ générique;
- ◆ L'état concurrent LockMgr indique si le contrôleur est libre de recevoir une commande. Lorsque le contrôleur démarre lui-même ou traite une commande il est mis à la valeur «Busy». Lorsqu'il est «Idle» et capable de recevoir des commandes, le LockMgr est mis à la valeur «Free».

3.2.4.7 Signalisation d'erreurs dans l'hierarchie du Run Control

L'état concurrent «Cfault » indique s'il y' a un problème avec un ou plusieurs des contrôleurs fils faisant partie actuellement de l'hierarchie contrôler par un contrôleur particulier (membership= In»). Il est prévu que l'événement erreur que prend le cluster RCFault dans l'état «Bad» sera publié sous les circonstances suivantes :

- ◆ Un enfant contrôleur est dans l'état «Bad»;
- ◆ Un enfant contrôleur est mort;
- ◆ Un timeout se produit pendant l'attente de la réponse d'un fils (durant la séquence d'initialisation ou durant l'envoi de la commande à l'enfant);

- ◆ Un ou plusieurs fils sont dans un état différent de l'état du parent;
- ◆ Un ou plusieurs fils sont occupés («Busy»).

L'événement «clear» que prend le cluster RCFault dans l'état «OK» sera publié quand le contrôleur détecte, à partir de la base dynamique du RC, que tous ses fils actuellement inclus dans l'hierarchie de Run Control (MemberShip= «In») sont en vie, dans le même état que le parent et libres («Free») pour accepter des commandes.

3.2.5 Message Reporting System (MRS)

Le but du Message Reporting System est de fournir la possibilité à tous les composants logiciels dans le DAQ et ses processus adjacents de signaler et de distribuer les messages d'erreurs et des messages d'informations aux autres composants du système distribué DAQ.

Le MRS devrait donner un moyen uniforme de distribution et de réception de messages d'erreurs à travers le système. Cependant, le MRS n'est pas un outil de communication générale. Les ressources du MRS fourniront des moyens en forme de librairie à un programme expéditeur (source) d'indiquer des messages d'erreurs uniques et à un programme récepteur (destinataire) d'obtenir des messages du MRS. Un moyen de définir des messages d'erreurs uniques dans le système et d'en faire usage devra être donné.

Le MRS est aussi responsable du filtrage des messages d'erreurs qui circulent entre les composants. Le système MRS permettra d'ajouter des attributs optionnels et des attributs obligatoires qui seront passés dans le message. Les applications auront la possibilité de redéfinir des groupes de messages qu'elles peuvent recevoir ou envoyer sans avoir à stopper ou à se déconnecter du MRS. Il donne également aux applications le moyen d'ajouter aux messages des qualificatifs optionnels ou obligatoires tout en ajoutant automatiquement lui-même certains qualificatifs aux messages. Un niveau de sévérité obligatoire sera défini soit pendant la définition du message ou soit par les applications qui l'ajoutent ensuite au message. Le MRS doit fournir à un récepteur les moyens de déterminer l'origine du message qu'il transporte, de récupérer les paramètres du message et le contenu. Pour optimiser le trafic de la communication sur le réseau les applications peuvent également sans devoir se déconnecter du MRS filtrer les messages qu'elles recevront dans le système DAQ selon les requêtes des utilisateurs ou définies par les experts du DAQ.

3.2.5.1 Vu d'ensemble

Les processus du DAQ qui exigent un système de transport de messages d'erreurs peuvent être à la fois sources et récepteurs de messages. Une application source de messages d'erreurs, une fois connectée au MRS, peut injecter un ou plusieurs messages dans le MRS à tout moment. Une application qui souhaite recevoir des messages doit se souscrire à un groupe de messages suivant le critère de souscription aux messages. Elle recevra alors un ou plusieurs messages qui rempliront les critères de souscription. Le traitement peut consister de quelques chose de simple: le login dans un fichier de messages ou sur le terminale de l'écran afin de procéder à l'analyse des erreurs. Pour filtrer les messages une commande de filtrage doit être initialisée à partir d'un commandeur. Un aperçu général est présenté dans la figure 3.6 ci-dessous.

L'un des objectifs visé par le design du MRS est de pouvoir définir une application interface simple à comprendre, à utiliser et qui doit être utilisable par tous les processus impliqués ou

relatifs au DAQ. Ainsi il doit pouvoir s'exécuter sur toutes les plates-formes en usage dans le système DAQ.

Le MRS doit fonctionner indépendamment de l'état du système DAQ. Un fichier log est également généré qui peut être consulté par les utilisateurs pour accomplir des analyses d'erreurs les plus approfondies si nécessaire.

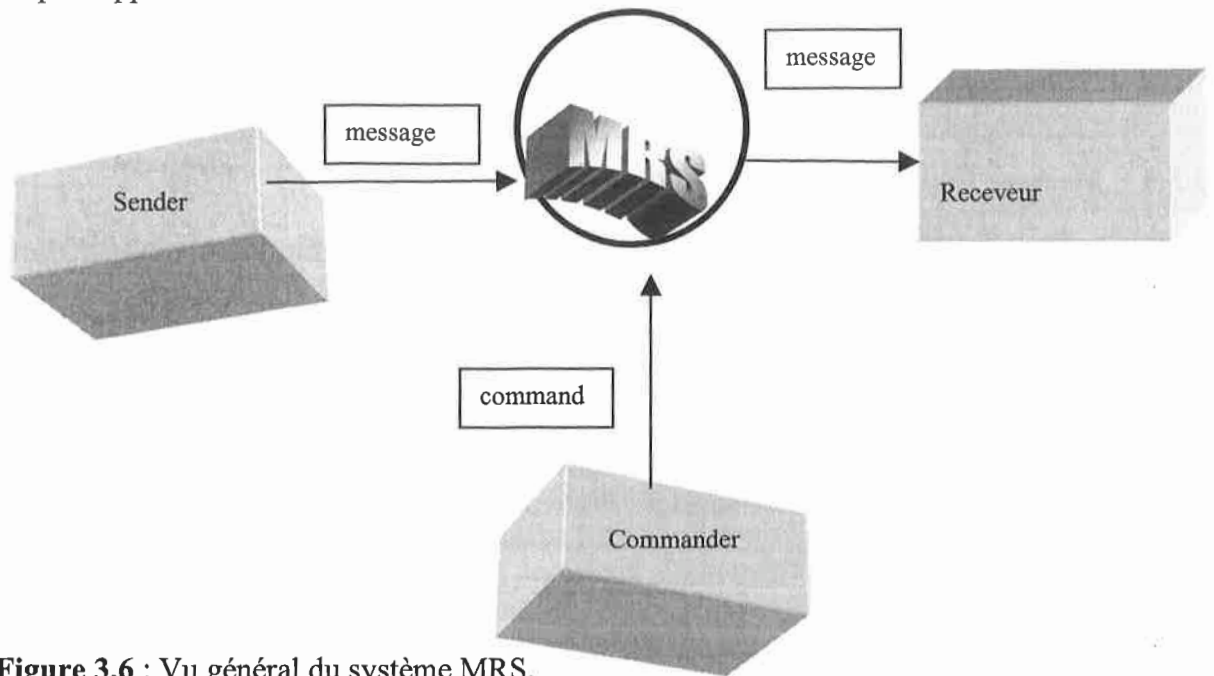


Figure 3.6 : Vu général du système MRS.

3.2.5.2 Structure du MRS

Le MRS est composé principalement de cinq modules : une interface d'envoi de messages (MRS sender application interface), un serveur du MRS (MRS server), une interface de réception de messages (MRS receiver application interface), une interface de commandes de filtrage de messages (MRS command interface) et une base de données du système MRS (MRS database).

Dans le système DAQ, un grand nombre d'utilisateurs doit accéder au système MRS. Cela impose une architecture hiérarchisée interne à MRS, qui n'est pas visible aux applications des utilisateurs. Les fonctions du serveur du MRS sont divisées en un serveur public du MRS et un nombre de serveurs privés. Le nombre de serveurs privés installés peut être ajusté suivant la taille et la complexité du système DAQ et les messages qui y circulent. Chaque serveur privé peut recevoir des messages d'un groupe de sources (sender) donnés, qui peut être logiquement ou géographiquement défini et peut varier suivant le trafic. Une fois démarrés, les serveurs privés joignent les serveurs publics en les envoyant un handler (leur propre handler) qui peut être utilisé plus tard pour accéder aux méthodes des serveurs privés du MRS.

Une application qui souhaite signaler des messages doit se connecter à un serveur public du MRS en utilisant le module qui assure l'interface d'envoi de message. Le serveur public du MRS choisit le serveur privé qui sera en charge de cette application. Il envoie alors un handler à l'interface MRS sender qui est utilisé pour les opérations de transport de messages. Donc le message sera envoyé directement au serveur privé du MRS où le message final est composé en ajoutant les informations correspondantes se trouvant dans la base de données. Dans le cas où le serveur MRS n'est pas présent dans le système, le MRS sender écrit le message dans un fichier de login.

Une application qui souhaite recevoir des messages doit se souscrire auprès du serveur publique à l'aide de l'interface de réception MRS. La souscription inclut la requête de réception de messages et le critère de souscription. Le serveur publique transmet alors cette souscription à tous les serveurs privés. Suivant leurs listes de souscription, tous les serveurs privés transmettent les messages correspondants directement à l'interface de réception. Quand une application veut se désinscrire à un groupe de messages particuliers, il envoie cette requête de la même façon au serveur publique, qui la transmet à tous les serveurs privés du MRS.

Un expert de DAQ peut initialiser les fonctions MRS en utilisant l'interface de commande du MRS. Un nombre de commandes peut être envoyé. Les messages peuvent être filtrés à l'aide de certaines commandes, à travers les serveurs privés. Les critères de filtrage sont composés de la même manière que les critères de souscription. Si un message remplit les critères de filtrage, il ne sera pas distribué par le MRS, même s'il remplit le critère de souscription. L'interface d'envoi envoie la commande de filtre au serveur publique qui la transmet aux serveurs privés où l'opération de filtrage est effectuée. La valeur du filtre courant peut être retrouvée par une commande d'obtention de la valeur du filtre courant avant de la changer. Toutes ces fonctionnalités sont disponibles dans l'interface de commandes du MRS. Le MRS peut également mettre à jour les informations internes de la base de données du MRS en utilisant l'interface de commandes du MRS. Il est également prévu de fournir un recouvrement des erreurs internes du MRS.

3.2.5.3 Définition de messages

Le MRS compose les messages à partir des informations qu'il a reçues de trois sources d'informations. Il existe une instance spécifique de l'information et des données dans la base de données de définition de messages. Un message qui est envoyé à MRS doit avoir été enregistré dans la base de données des messages. Les informations d'identification dans le système et dans le temps sont seulement ajoutées par l'interface d'envoi du MRS lorsque le message est distribué.

Le message tel qu'il est envoyé par l'application source est composé du message ID, des paramètres exigés, des paramètres optionnels et des qualificatifs.

L'interface MRS sender API ajoute automatiquement des qualificatifs pour identifier le message uniquement dans le système.

Le message final, construit par le MRS au niveau du serveur privé, est composé de message ID, du texte de message, des paramètres exigés, des paramètres optionnels, des qualificatifs automatiquement ajoutés et les qualificatifs optionnels. Les éléments d'un message :

Message ID : Le message ID est l'identificateur de message tel qu'il est défini dans la base de données de message IDs.

Message texte : Le message texte est le texte ascii tel qu'il est défini dans la base de données de message IDs.

Paramètres : Dans la base de données de définition du message, le message peut optionnellement être défini afin qu'il contienne une place pour garder un ou plusieurs paramètres. Dans ce cas la valeur réelle du paramètre sera fournie par celui qui fait la distribution de ce message.

Qualificateurs : Les qualificateurs sont des pièces d'informations du message, mais ne représente pas son contenu (c'est-à-dire l'identification de l'expéditeur).

Les qualificateurs automatiquement ajoutés : Les qualificateurs qui sont automatiquement ajoutés par l'interface d'envoi du MRS :

- Le nom du programme;
- Le nom de partition;
- Le nom de la machine;
- Instance;
- La date;
- Le temps.

Niveau de sévérité des qualificateurs : Un qualificateur définit le niveau de sévérité d'un message par défaut tel qu'il est défini dans la base de données. L'utilisateur peut donc définir son propre niveau de sévérité. Le niveau de sévérité est converti de façon interne en qualificateur.

Les différents niveaux de sévérités sont : FATAL, ERROR, WARNING, INFO, SUCCESS.

Les qualificateurs optionnels : Ils sont principalement utilisés pour décrire le contexte du message et peuvent être ajoutés au message de deux façons : l'application source du message peut ajouter un ou plusieurs qualificateurs; un ou plusieurs qualificateurs peuvent aussi être ajoutés à la définition du message défini dans la base de données.

3.2.5.4 Critère de souscription

L'application receveur doit fournir l'information qui permet au MRS d'accomplir ses tâches de signalisation et distribution de messages d'erreurs. Cela est fait par les critères de souscription. Chaque application doit définir une combinaison de ces critères lorsqu'elle doit faire une souscription auprès du MRS. Les possibilités de souscription sont des expressions logiques de : message ID, qualificateurs, négation de qualificateurs ou ID et la réception de tous les messages.

Les mots réservés sont : AND, OR, NOT, ALL.

3.2.5.5 Assemblage de message

Un message est construit à partir des informations soumises par l'utilisateur à l'interface MRS sender, des informations ajoutées par l'interface MRS sender et des informations de la base de données du serveur MRS. Le message final est composé en fonction de la requête de l'utilisateur, par l'interface de réception du MRS.

Exemples :

1) L'exemple suivant représente un message et son contenu :

L'utilisateur source fournit l'ID et les paramètres exigés :

◆ RCL_START 27.

L'interface d'envoi du MRS ajoute le nom de la partition, le nom du programme, le nom de la machine, l'instance, la date et le temps :

◆ RCL_START 27 MYPART RC suds06.cern.ch 18722 1997/03/30 16 :08 :06

Le serveur du MRS ajoute l'information de définition du message de la base de donnée, qui est le niveau de sévérité, le texte du message, le format (la référence) du paramètre exigé et un qualificateur optionnel :

- ◆ RCL_START 27 MYPART RC suds06.cern.ch 18722 1997/03/30 16 :08 :06
INFO Run number %d has started BE.

En fonction de la requête de l'utilisateur, le message final est composé par l'interface de réception du MRS comme suit :

- ◆ RCL_START MYPART RC suds06.cern.ch 18722 1997/03/30 16 :08 :06 INFO
BE Run number 27 has started.

2) Le même exemple peut être étendu :

L'utilisateur source fournit l'ID, le niveau de sévérité, un paramètre et un qualificateur optionnel:

- ◆ RCL_START WARNING 27 CALTEST.

L'interface d'envoi du MRS ajoute le nom de la partition, le nom du programme, le nom de la machine, l'instance, la date et le temps :

- ◆ RCL_START WARNING 27 CALTEST MYPART RC suds06.cern.ch 18722
1997/03/30 16 :08 :06.

Le serveur MRS ajoute l'information de la base de données de définition de message qui est le message texte, le format pour les paramètres obligatoires, pas de niveau de sévérité, et un qualificateur optionnel :

- ◆ RCL_START WARNING 27 CALTEST MYPART RC sud06.cern.ch 18722
1997/03/30 16 :08 :06 Run number %d has started BE.

Selon la demande de l'utilisateur, le message final est composé par l'interface de réception :

- ◆ RCL_START WARNING CALTEST MYPART RC suds06.cern.ch 8722
1997/03/30 16 :08 :06 BE Run number 27 has started.

3.2.5.6 Filtrage des messages

Les messages peuvent être filtrés. Les critères de compositions des filtres sont les mêmes que les critères de compositions de souscription. Les fonctions de filtrage sont disponibles dans l'interface de commandes du MRS. Avant de poser un nouveau filtre, il est recommandé de lire d'abord le filtre courant et de composer le nouveau critère à partir de l'expression de ce filtre courant et des nouvelles informations à disposition. L'avantage du filtrage est de désengorger le trafic de communication du réseau. Un message qui satisfait un critère de filtre ne sera jamais transporté par le MRS même s'il satisfait les critères de souscription.

3.2.5.7 Les fonctions de contrôle du fichier login

Le serveur crée un fichier log dans lequel il répertorie toutes ses actions. Le nom de ce fichier log est MRS<Partitionname>.serveur.log. Le niveau des informations du fichier log peut être mis à jour grâce à des commandes à cet effet. Ils existent trois niveaux d'informations dans le logfile : les informations moins détaillées, les informations assez détaillées et les informations vide.

3.3 Conclusion

Le Back-end software gère et contrôle tous les sous-systèmes du DAQ. Il est composé de cinq principaux composants : la base de données de configuration, le PMG, le RC, le MRS et l'IS.

La base de données de configuration permet de paramétrer, de configurer et de décrire le système d'architecture hardware et software du DAQ.

Le PMG est un système distribué du DAQ qui permet de créer, de lancer de stopper et de contrôler les processus des composants sur les stations de travail du DAQ.

L'IS est un système qui permet l'échange de l'information dans le DAQ en rendant disponible le contenu de celle-ci à un ou plusieurs groupes de composants destinataires dans le DAQ.

Le RC a pour responsabilité de contrôler et coordonner les opérations des sous-systèmes du DAQ. Représenté par une machine à état fini CHSM, il a une structure d'arborescente.

Le MRS permet à tous les composants softwares la signalisation et la distribution des messages d'erreurs dans le DAQ. Pour recevoir un message d'erreur, l'application ou le composant receveur doit procéder à une souscription auprès du MRS. Le MRS permet également le filtrage des messages qui circulent dans le système et la génération de logfile.

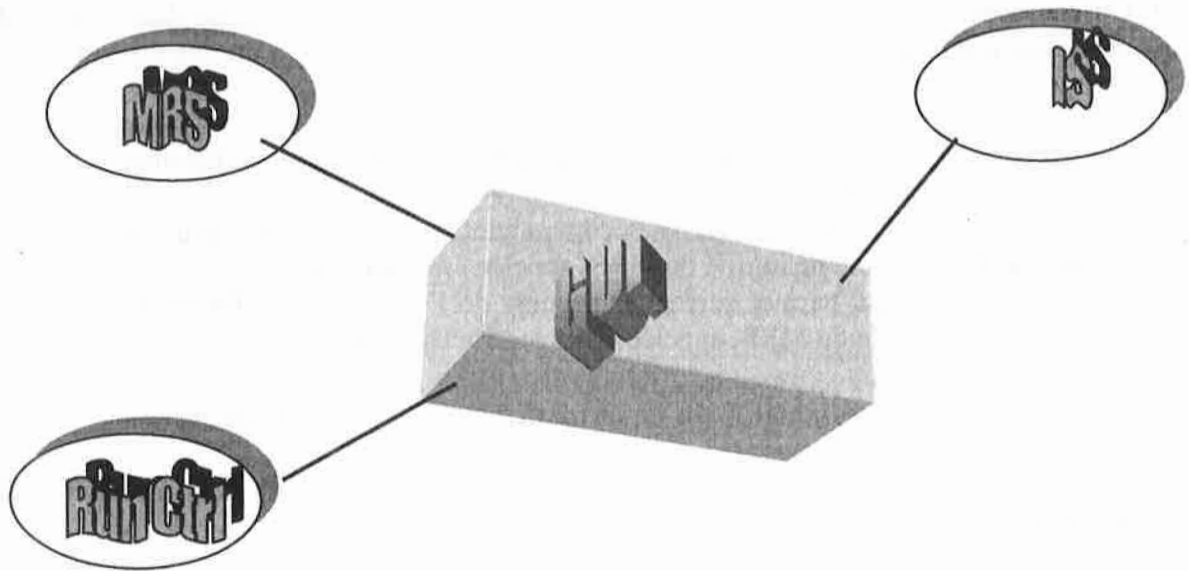
Ces cinq composants cités ci-dessus font du Back-end un rassembleur de tous les sous-systèmes du DAQ.

4. L'Interface Graphique Générale Utilisateur ou Graphic User Interface (GUI)

4.1 Vu d'ensemble

La contribution apportée par ce projet de mémoire à GUI s'intéresse à l'interaction entre les composants l'IS, le RC et le MRS au sein du système global Back-end. Rappelons également les cinq composants qui constituent le pilier sur lequel tous les autres sous-systèmes du DAQ se reposent pour coopérer et accomplir leurs tâches : l'IS, le MRS, le RC, le PGM et le DAQ Supervisor. On se penche dans ce travail sur comment représenter dans l'interface graphique général GUI, les interactions entre l'IS, le RC et le MRS et comment intégrer ensemble dans le GUI, les interactions entre ces trois composants.

La méthodologie de travail retenue au CERN pour réaliser ce GUI est que la version finale du GUI est au niveau de Monsieur Moneta Lorenzo qui est le co-superviseur. Les différents modules du GUI sont testés et validés d'abord avant de les intégrer dans la version finale du GUI. En ce qui concerne les parties confiées à ce projet de diplôme, nous avons d'abord construit une version de teste du GUI que nous avons appelé Rc_displayfinal qui intégrera les différents modules au fur et à mesure qu'ils sont construits. Ce n'est qu'après leur teste et leur validation dans cette version de teste qu'ils sont ensuite transférés dans la version finale du GUI. La figure 4.1 représente un aperçu général du le travail accompli.



La figure 4.1 : Aperçu général du contexte du GUI dans ce travail.

Deux composants softwares ont été construits dans ce projet de mémoire qui permettent de représenter et d'intégrer dans le GUI toutes les interactions entre les composants softwares l'IS, le RC et le MRS. Chacun des deux composants accomplit un ensemble de fonctionnalités unique résultant des interactions entre l'IS, le RC et le MRS.

Ces deux composants sont :

- 1) Le composant appelé «GUI_RC Interface», qui prend en charge toute la représentation graphique des informations pertinentes qui résultent des interaction entre l'IS, le RC et les

applications externes ou auxiliaires au système Back-end. Il est composé des sous-composants suivants :

- ◆ Le sous-composant «Display Information» constitué également des parties suivantes :
 - Un GUI (expert) appelée *RcCommands* qui permet d'envoyer des commandes expertes et non expertes à tous les contrôleurs;
 - Un GUI appelé *RcStatusDisplay* qui permet d'afficher les différents statuts des contrôleurs;
 - Une GUI appelé *RcErrorCommand* qui permet de gérer le membership de la structure arborescente de RC et les erreurs;
 - Deux classes qui s'occupent de la gestion des partitions et le chargement des contrôleurs;
 - ◆ Un sous-composant «GUI_RC_IS Communication» qui fait office de passerelle de communication d'une part entre les sous-modules cités ci-dessus et le composant RC, et d'autre part entre ces mêmes sous-modules et le composant IS. Il est aussi constitué des sous-modules suivants :
 - Une interface d'envoi de commandes et d'informations entre le GUI et le RC appelé *RcSendCommands*;
 - Une interface de réception appelée *RcReceiveCommands* qui permet de recevoir toutes les informations que l'IS délivre, de les traiter avant de les distribuer dans le GUI;
 - Une interface de notification d'informations;
- 2) Un composant «GUI_MRS Interface» qui s'occupe de la représentation dans le GUI des messages d'erreurs, du filtrage des messages et de la souscription auprès du MRS. Il est composé des sous-composants suivants :
- Un GUI qui permet l'affichage formaté et compréhensible par l'utilisateur des messages du MRS, appelée *MrsTable*;
 - Un GUI qui permet d'envoyer des commandes expertes à MRS pour la gestion des filtres et le contrôle de logfile du MRS, appelée *MrsCmdGui*;
 - Une interface qui permet suivant la requête de l'utilisateur d'afficher l'un des trois formats de logfile du MRS, appelée «Mrs logfile Display».

4.2 Analyse et modélisation du composant « GUI_RC Interface »

Dans cette section, on donne quelques exemples de l'analyse et de la modélisation qui ont été faites sur le «GUI_RC Interface».

4.2.1 Use case d'un scénario de lookup pour rc_handler et un is_handler

Il s'agit d'un scénario de recherche de deux instances rc_handler et is_handler de deux objets différents respectivement publiés par le RC et l'IS, pour l'envoi de commandes et la réception d'informations.

Ces «use cases» sont résumés comme suit :

A. Keeper Partition List

- ◆ L'objet (class) *Rc_dispayfinal* remplit sa liste de partition par les noms des partitions disponibles.

B. Choose Partition

- ◆ L'utilisateur choisit un nom de partition dans la liste de partitions et valide son choix;
- ◆ La classe *Rc_displayfinal* affiche le nom de partition choisit et considère que c'est la partition de travail;
- ◆ L'objet *Rc_displayfinal* envoie une requête à la classe *ListPartitionAndContr* pour remplir sa liste de contrôleurs.

C. Load Controllers List

- ◆ L'objet *ListPartionAndContr* envoie une requête à l'objet *rdbInterface* pour avoir la liste des contrôleurs dans la base de données rdb;
- ◆ L'objet *rdbInterface* charge la liste de contrôleurs disponibles dans la partition qui a été choisie et la renvoie à l'objet *ListPartionAndContr*;
- ◆ L'objet *ListPartionAndContr* remplit son tableau prévu à cet effet.

D. Keeper Controllers List

- ◆ L'objet *Rc_display* remplit sa liste de contrôleurs à partir du tableau de l'objet *ListPartionAndContr*.

E. ChooseController

- ◆ L'objet *Rc_displayfinal* affiche l'arborescence des contrôleurs disponible;
- ◆ L'utilisateur choisit un contrôleur dans l'arborescence affichée par *RC_displayfinal*.

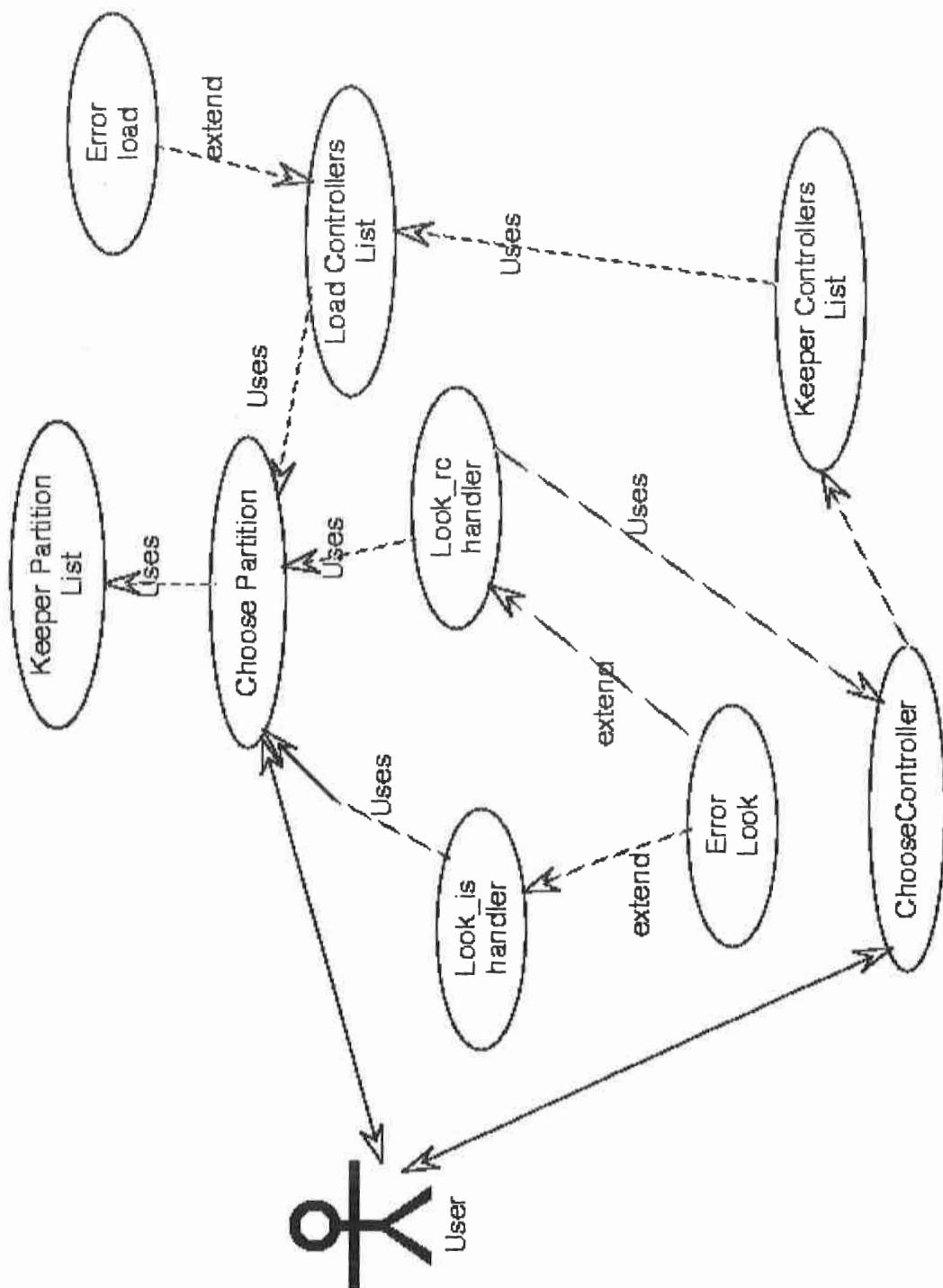
F. Look_rc handler

- ◆ *Rc_displayfinal* demande à l'objet *RcSendCommands* de chercher (lookup) le *rc_handler*, l'instance de l'objet publié par le serveur RC, qui doit être utilisé pour envoyer les commandes à RC;
- ◆ *RcSendCommands* poste un lookup (recherche), pour trouver l'instance de cet objet.

G. Look_is handler

- ◆ *Rc_displayfinal* demande à l'objet *RcReceiveCommands* de chercher (lookup) l'instance *is_handler* de l'objet publié par le serveur de l'IS, qui sera utilisé pour faire la souscription auprès de l'IS et de recevoir les nouvelles informations venant de l'IS;
- ◆ *RcReceiveCommands* recherche (lookup) cette instance.

Le diagramme 4.1 représente le use case du scénario exposé ci-dessus.



looking_is_rc_handler on 10/22/1999

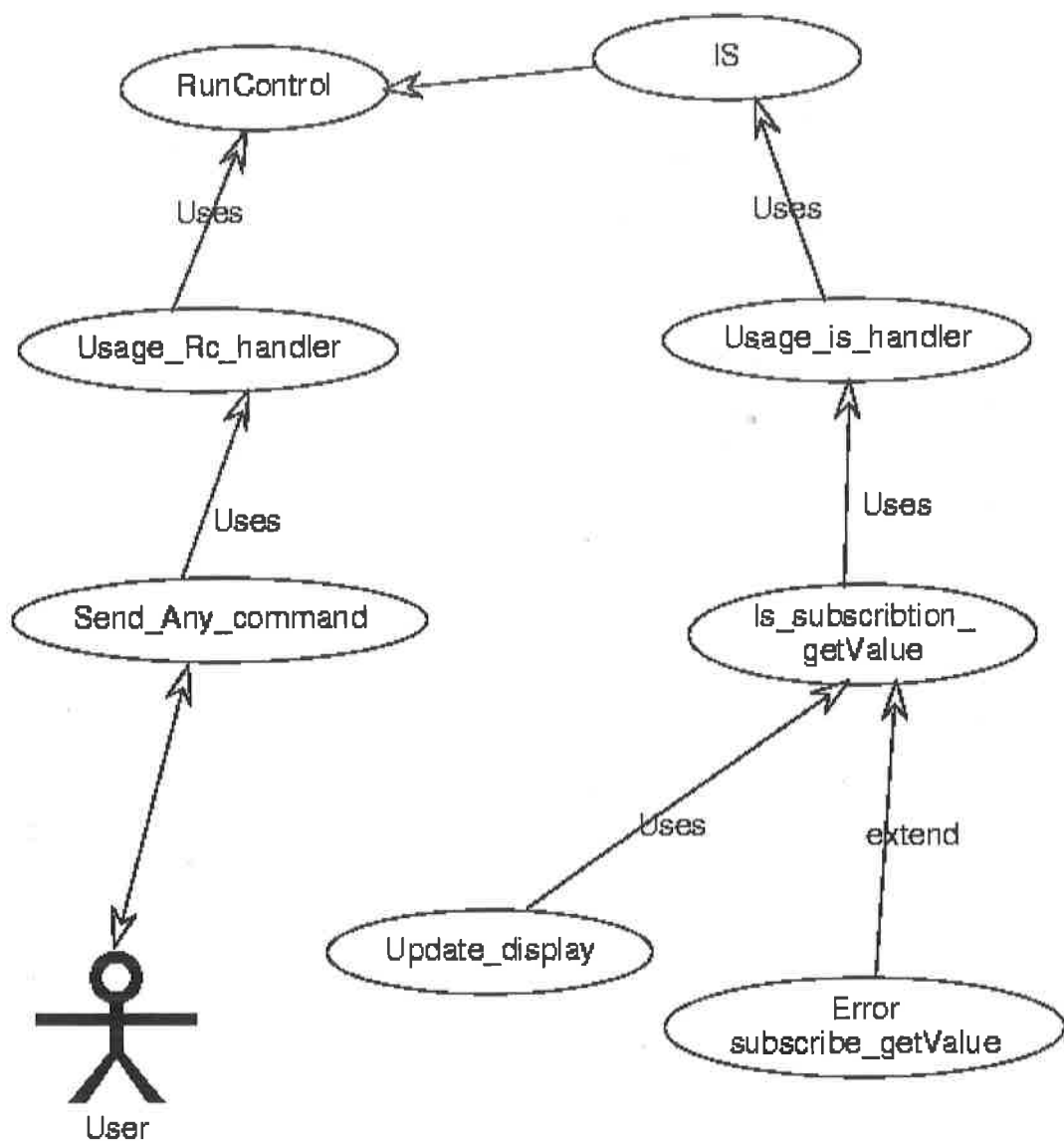
Diagramme 4.1 «use case» *is_rc_handler*.

4.2.2 Use case pour un scénario d'envoi de commandes et de réception de nouvelles informations

Dans ce scénario, il s'agit de l'utilisation des instances *rc_handler* et *is_handler* pour envoyer une commande et recevoir des informations. Ces «use cases» sont résumés comme suit :

- A. **Usage_rc_handler**
 - ◆ *RcSendCommand* retourne l'instance *rc_handler* à la classe *Rc_displayfinal* pour d'éventuel envoi de commandes.
- B. **Usage_is_handler**
 - ◆ *RcReceiveCommands* retourne l'instance *is_handler* à la classe *Rc_displayfinal* pour d'éventuelle souscription auprès de l'IS.
- C. **Is_subscription**
 - ◆ *Rc_displayfinal* effectue une souscription auprès de l'IS pour recevoir de nouvelles informations.
- D. **Send_Any_command**
 - ◆ L'utilisateur clique sur une commande d'une des deux objets *RcCommands* ou *RcErrorCommands*;
 - ◆ *RcCommands* utilise le *rc_handler* de la classe *RcSendCommands* pour envoyer la commande qui a été cliquée.
- E. **Update_display**
 - ◆ Une commande doit avoir été envoyé par *RcSendCommands*;
 - ◆ L'objet *TrueCall* est automatiquement invoqué par l'IS pour notifier à l'objet *RcReceiveCommands* l'arrivée de nouvelles informations;
 - ◆ *RcReceiveCommands* met à jour les informations de la classe *RcCommands*;
 - ◆ *RcReceiveCommands* met à jour les informations de la classe *RcStatusDisplay*;
 - ◆ *RcReceiveCommands* met à jour les informations de la classe *RcErrorCommands*.

Le diagramme 4.2 représente le «use case» de ce scénario.



Send_Command_And_display_Info on 3/1 2/1999

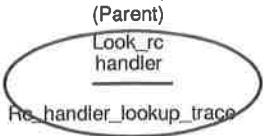
Diagramme 4.2 «Use case» pour Send_Command_And_display_Info

4.2.3 Event Trace de Use Case “*Look_rc_handler*”

Dans le premier scénario, la decomposition du use case “*Look_rc_handler*” fournit le flux d’événements et l’interaction entre les classes qui participent à ce scénario. Dans ce qui suit nous montrons la sequence du flux d’événements qui se produit dans l’interaction entre un utilisateur et les classes qui participent a ce scénario. Le diagramme 4.3 montre ce “event trace”.

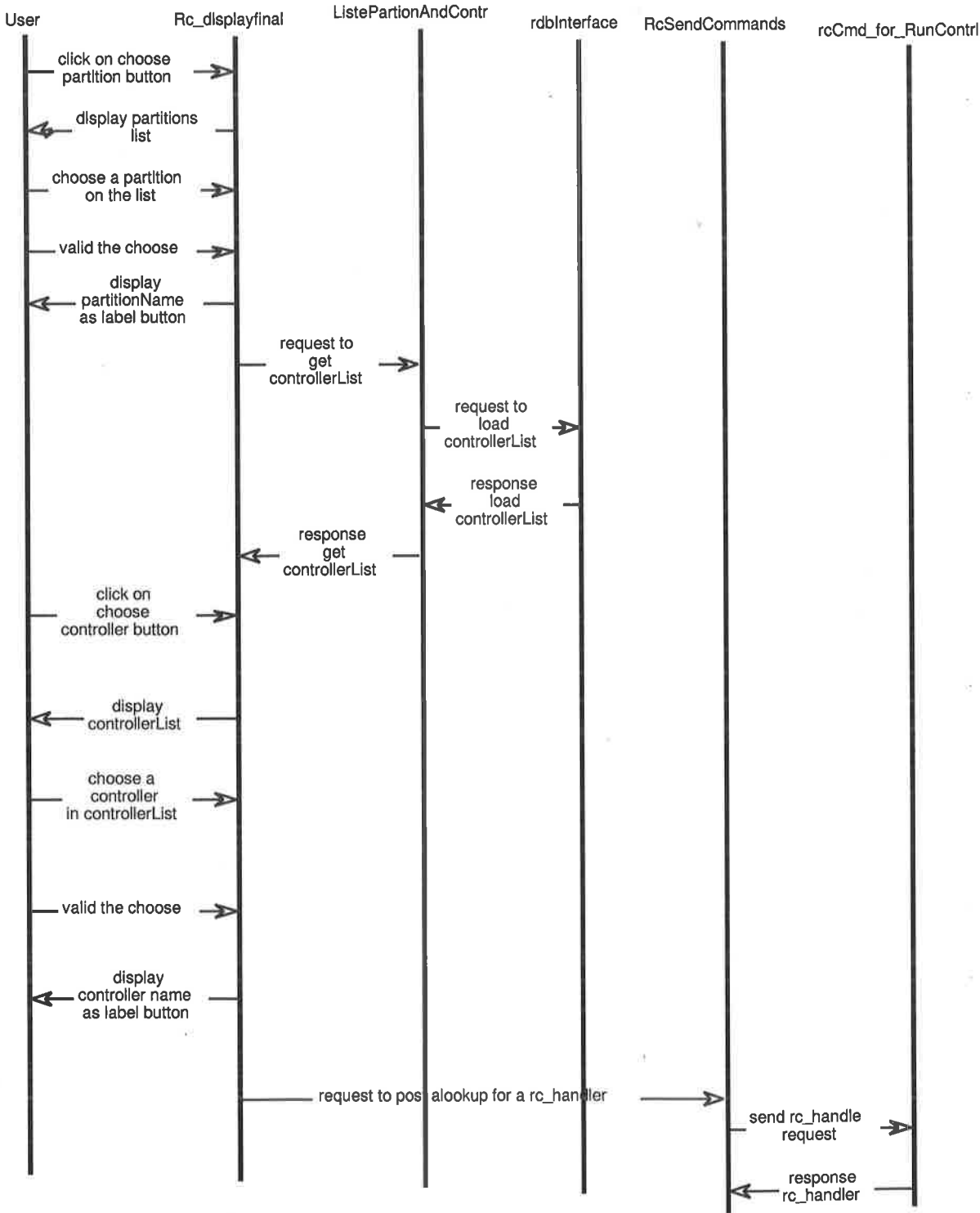
- 1- l'utilisateur demande l'affichage de la liste de partitions;
- 2- la classe *RC_displayfinal* lui renvoie la liste de partitions;
- 2- l'utilisateur choisit une partition dans la liste
- 3- l'utilisateur valide son choix;
- 5- La classe *Rc_displayfinal* retient le nom de cette partition et considère pour le reste de session qu'elle est la partition de travail jusqu'une nouvelle demande de changement de partition lui arrive;
- 6- La classe *Rc_displayfinal* demande à la classe *ListPartionAndContr* de préparer sa liste de contrôleurs;
- 7- La classe *ListPartionAndContr* envoie une demande de chargement des contrôleurs à la classe *rdInterface*;
- 8- La classe *rdInterface* charge les contrôleurs de la partition choisies par l'utilisateur depuis la base de données rdb et retourne la liste de ces contrôleurs à la classe *ListPartionAndContr* qui lui en a fait la demande;
- 9- La classe *ListPartionAndContr* remplit sa liste et retourne une instance de cette list à la classe *Rc_displayfinal* qui lui avait demander de préparer la liste de controleurs;
- 10- L'utilisateur demande l'affichage de la liste de controleurs à la classe *Rc_displayfinal*;
- 11- *Rc_displayfinal* affiche la liste de contrôleurs qu'elle vient de recevoir;
- 12- l'utilisateur choisit un nom de controleur dans la liste;
- 12- l'utilisateur valide son choix;
- 13- La classe *Rc_displayfinal* retient le nom du controleurs dans l'arborescence des contrôleures;
- 14- La classe *Rc_displayfinal* demande à la classe *RcSendCommands* de rechercher sur le réseau une instance de l'objet publié par RC dont le handler est *rc_handler*;
- 15- La classe *RcSendCommands* recherche l'objet en question auprès du server du composant RC;
- 16- La classe *Rc_displayfinal* obtient une instance de cet objet qui lui permettra d'envoyer des commandes.

Diagramme 4.3 «Event trace» pour la recherche du rc_handler.



EVENT TRACE DIAGRAM TO POST A LOOKUP FOR A rc_handler

The classes that participe to this scenario are follows:
Rc_displayfinal, ListPartionAndContr, rdbInterface, RcSendCommands.



Look_rc_handler__1 on 10/5/1999

4.2.4 Event Trace pour le Use Case "Update_display"

On décompose le Use Case "Update_display" pour construire son diagramme event Trace pour obtenir la séquence du flux d'événements dans les interactions entre les classes qui participent à ce use case pour un scénario d'envoi d'une commande et de mise à jour de nouvelles informations. Notons que la mise à jour des informations par le *Rc_displayfinal* peut être provoquée par n'importe quel autre programme outre qu'un utilisateur externe. Pour étudier cet event trace, nous choisissons un cas particulier où un utilisateur choisit une commande dans la classe *RcCommands* et l'envoie:

- 1- L'utilisateur clique sur une commande dans l'interface graphique de la classe *RcCommands*;
- 2- la classe *RcCommands* transmet la commande cliquée à la classe *RcSendCommands*;
- 3- La classe *RcSendCommands* envoie la commande qu'elle a reçue, à RC par l'intermédiaire de l'instance *rc_handler* qu'elle contient;

Remarque: Une fois que le RC reçoit la commande, Il renvoie le resultat de l'exécution de cette commande au composant l'IS, et l'IS distribue le resultat à toutes les applications qui se sont souscrites pour ce genre d'information.

- 4- Le composant IS notifie les nouvelles informations à tous ces clients qui ont fait une souscription à ce type d'information, en particulier à la classe *TrueCall*.
- 5- La classe *TrueCall* fait à son tour une notification de ces nouvelles informations à la classe *RcReceiveCommands*;
- 6- La classe *RcReceiveInfo* met à jour les informations de la classe *RcCommands*;
- 7- La classe *RcReceiveInfo* met à jour les informations de la classe *RcStatusDisplay*;
- 8- La classe *RcReceiveCommands* met à jour les informations de la classe *RcErrorCommands*.

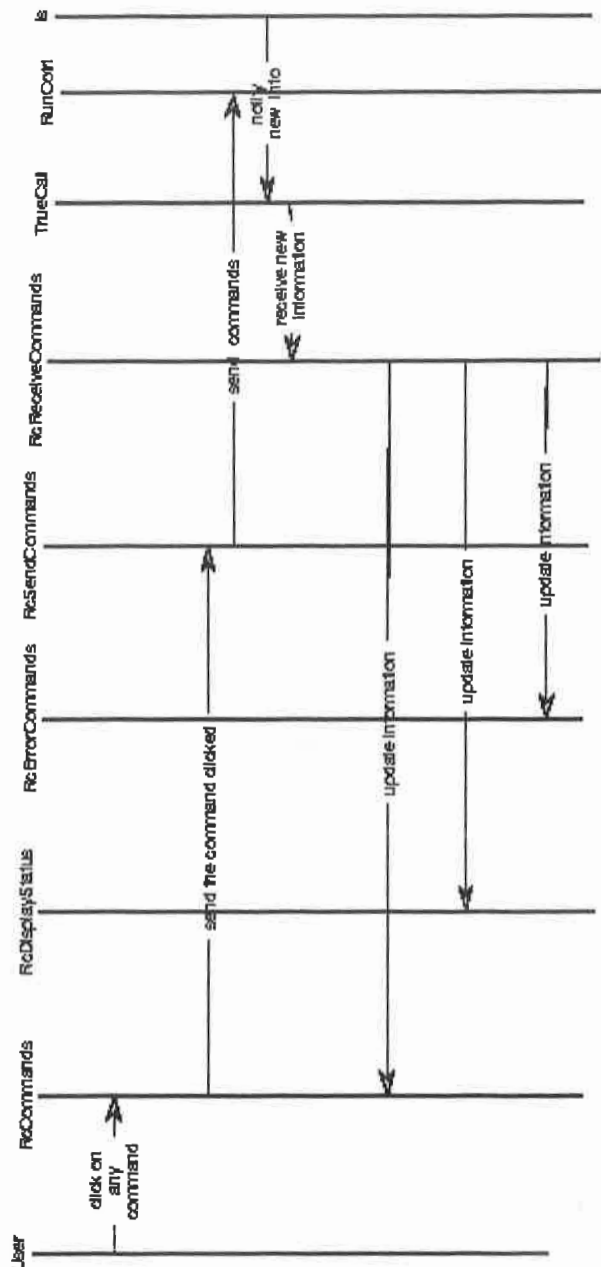
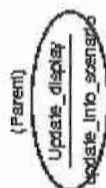
Le diagramme 4.4 illustre cet event trace.

EVENT TRACE DIAGRAM FOR UPDATE INFORMATION DISPLAY

This is an event trace diagram that allows you to show a scenario to send any command in ReCommands display and update ReCommands display with the new information that will be received.

The classes that participate scenario are follows:

ReCommands, ReDisplayStatus, ReErrorCommands, ReSendCommands, ReReceiveCommands, TrueCall.

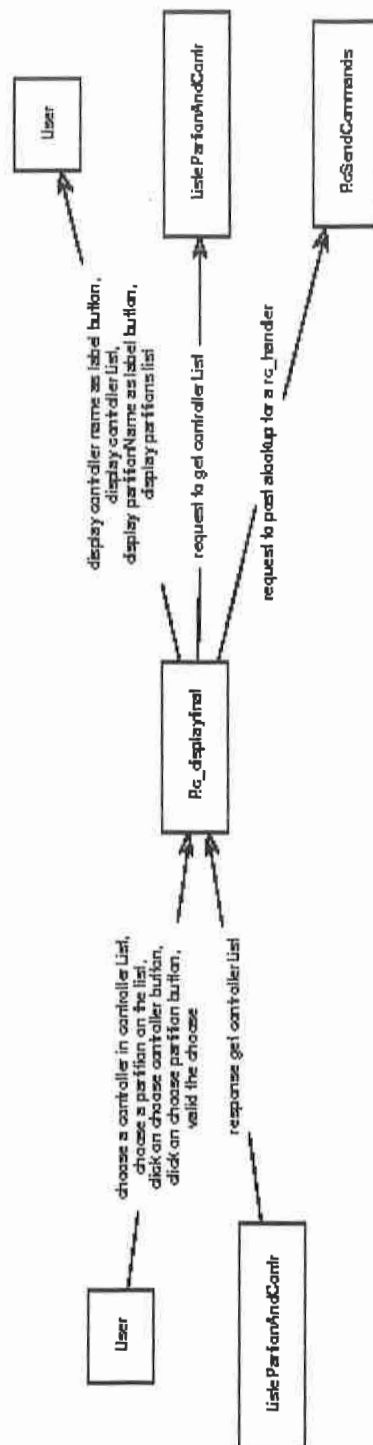


Update_display__1 on 3/13/1999

Diagramme 4.4 Event trace pour le use case pour Update_dispay

4.2.5 Event Summary diagramme pour la classe *Rc_displayfinal*

On peut établir un diagramme des event summary pour chaque classe participant à un diagramme de «event trace». Cela permet de voir le flux d'événements entre les classes dans le domaine modelisé. On peut sélectionner une classe dans un diagramme «event trace», pour voir quels sont les flux d'événements entrants et sortants de cette classe. On donne un exemple de «event summary pour la classe *Rc_display* choisie dans le diagramme de «event trace» du use case *Look_rc_handler*. Le diagramme 4.5 montre cet event sommmary pour la classe *Rc_displayfinal*.



Rc_displayfinal_events on 3/12/1999

Diagramme 4.5 "Event summary" pour *Rc_displayfinal*.

4.2.6 Modèle d'objet du composant "GUI_RC Interface"

Le diagramme 4.6 montre la modélisation des objets du "GUI_RC interface" qui met en évidence les relations d'héritages, les membres et d'associations qui existent entre les objets dans le domaine. Nous donnons une explication claire des arcs qui existent entre les classes dans ce diagramme:

- ◆ Entre la classe *Rc_displayfinal* et la classe *RcCommands*, il existe un arc qui symbolise une relation membre. C'est à dire que la classe *RcCommands* est une classe membre de la classe *Rc_displayfinal* de cardinalité égale à 1. Donc une instance active de la classe mère *Rc_displayfinal* contient exactement une instance de la classe *RcCommands*. La classe *Rc_displayfinal* est alors la classe contenante et la classe *RcCommands* est la classe contenue.
- ◆ Entre la classe *Rc_displayfinal* et la classe *RcStatusDisplay*, il existe également un arc qui symbolise une relation membre, qui veut aussi dire que la classe *RcStatusDisplay* est une classe membre de la classe *Rc_displayfinal*, de cardinalité 1. Donc une instance de *Rc_displayfinal* contient exactement une instance de *RcStatusDisplay*. Alors la classe *Rc_displayfinal* est la classe contenante et la classe *RcStatusDisplay* est la classe contenue.
- ◆ Le même type de relation membre avec la même cardinalité existe entre les classes *Rc_displayfinal* et *RcErrorCommands*.
- ◆ Entre la classe *Rc_displayfinal* et la classe *ListPartionAndContr*, il existe un arc qui symbolise une relation d'association de nom de *display_controllers*, de cardinalité des deux côtés de l'arc égale à 1. Dans cette association la classe *Rc_displayfinal* joue le rôle d'afficheur des contrôleurs *controllers_displayer* et la classe *ListPartionAndContr* joue le rôle de conteneur des contrôleurs *controllers_contener*.
- ◆ Entre la classe *Rc_displayfinal* et la classe *RcSendCommands*, il existe une relation d'association de nom de *lookup rc_handler* de cardinalité égale à 1 des deux côtés de l'arc. L'existence de cette association entre *Rc_displayfinal* et *RcSendCommands* est justifiée du fait qu'un objet *rc_handler* est publié par un serveur du composant RC dont la classe *Rc_displayfinal* et ces classes membres ont besoin pour envoyer des commandes, d'une part, et d'autre part comme ni la classe *Rc_displayfinal*, ni ses classes membres ne savent comment aller chercher l'instance *rc_handler* de cet objet publié, et que seulement la classe *RcSendCommands* sait la rechercher, alors la classe *Rc_displayfinal* et la classe *RcSendCommands* s'associe dans l'association "*lookup rc_handler*" pour accomplir ce service de recherche de *rc_handler*.
- ◆ Entre la classe *Rc_displayfinal* et la classe *RcReceiveInfo*, il existe une relation d'association de nom de *Lookup_is_handler_and_subscribe* de cardinalité égale à 1 des deux côtés de l'arc. L'existence de cette association est également justifiée du fait que, pour recevoir de nouvelles informations venants d'autres applications et composants du DAQ, chaque application dont notre *Rc_displayfinal* doit proceder à une souscription auprès du composant IS. A cet effet un des serveurs du composant IS publie un objet *is_handler* pour les applications qui desirent faire une souscription. Comme c'est la classe *RcReceiveInfo* qui sait comment aller rechercher (lookup) l'instance publiée de cet objet et sait faire la souscription, alors la classe *Rc_displayfinal* s'associe avec *RcReceiveInfo* dans

l'association *Lookup_is_handler_and_subscribe* pour accomplir ces services de recherche de *is_handler* et de souscription.

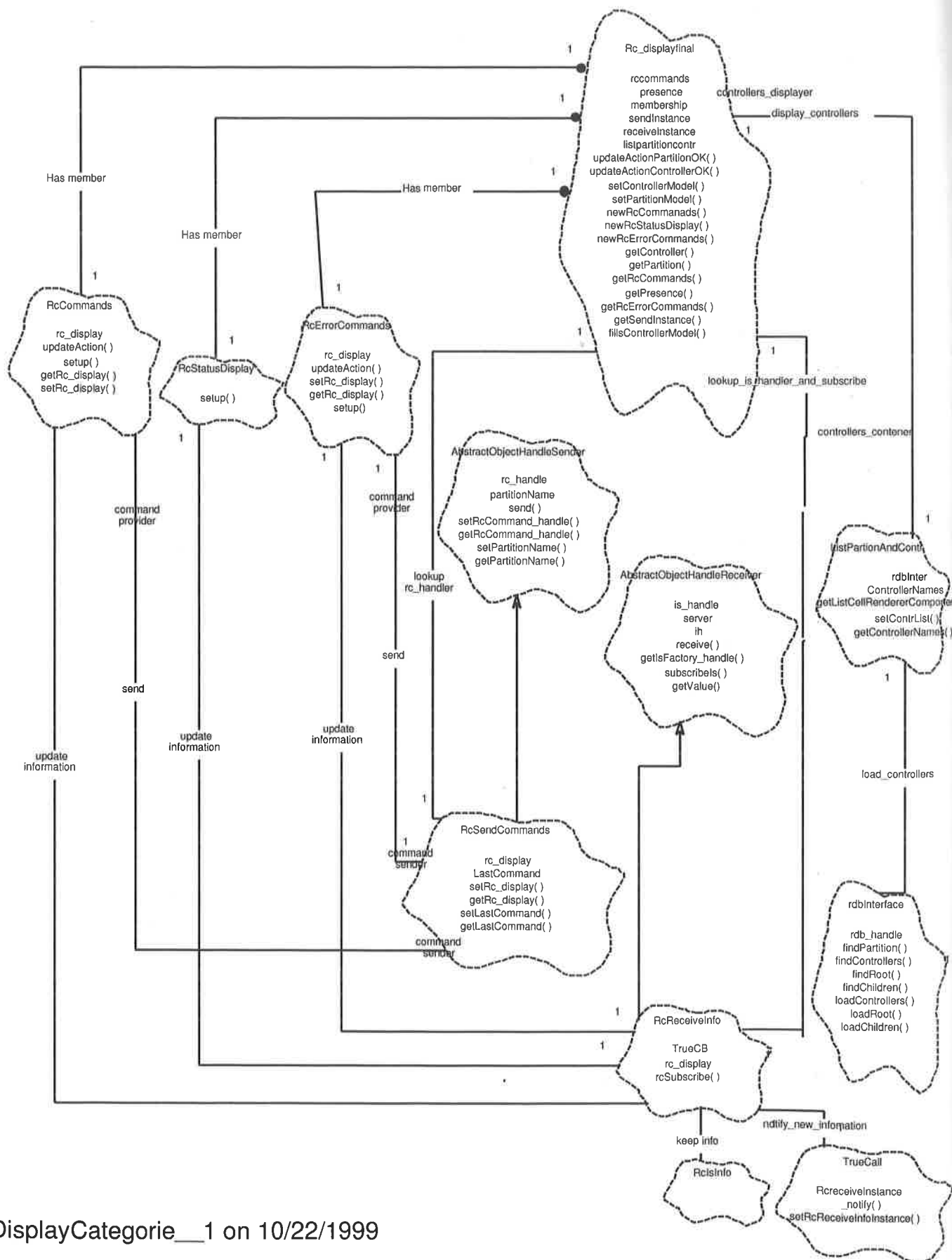
- ◆ Entre la classe *ListPartionAndContr* et la classe *rdInterface*, il existe une relation d'association de nom de *load_controllers* de cardinalité égale à 1 des deux côtés de l'arc. Dans cette association, la classe *ListPartionAndContr* a besoin de connaître les contrôleurs disponibles dans la partition de travail, et la classe *rdInterface* sait comment aller interroger la base de donnée *rd* pour obtenir ces contrôleurs. Ainsi elles s'associent pour accomplir ce service.
- ◆ Il existe une relation d'association entre la classe *RcCommands* et la classe *RcSendCommands* de nom de "send", de cardinalité égale à 1 des deux côtés de l'arc. Dans cette association la classe *RcCommands* joue le rôle de "command provider" et la classe *RcSendCommands* joue le rôle "command sender".
- ◆ Il existe une relation d'association entre la classe *RcErrorCommands* et la classe *RcSendCommands* de nom de *send* de cardinalité égale à 1 des deux côtés de l'arc, dans laquelle la classe *RcErrorCommand* joue le rôle de "command provider" et la classe *RcSendCommands* joue le rôle *command sender*.
- ◆ Il existe une relation d'héritage entre la classe *AbstractObjectHandleSender* et la classe *RcSendCommands* symbolisée par une flèche qui va de la classe *RcSendCommands* vers la classe *AbstractObjectHandleSender*, qui veut dire que la classe *RcSendCommands* hérite de la classe *AbstractObjectHandleSender*. Donc dans cette relation d'héritage, la classe *AbstractObjectHandleSender* est la super-classe et la classe *RcSendCommands* est la sous-classe.
- ◆ Il existe une relation d'héritage entre la classe *AbstractObjectHandleReceiver* et la classe *RcReceiveInfo* symbolisée par la flèche qui part de la classe *RcReceiveInfo* vers la classe *AbstractObjectHandleReceiver*, qui veut dire que la classe *RcReceiveInfo* hérite de la classe *AbstractObjectHandleReceiver*. Dans cette relation d'héritage la classe *AbstractObjectHandleReceiver* est la super-classe et la classe *RcReceiveInfo* est la sous-classe.
- ◆ Il existe une relation d'association entre la classe *TrueCall* et la classe *RcReceiveInfo* de nom de "notify_new_information" de cardinalité égale à 1 des deux côtés de l'arc. Grâce à cette association les nouvelles informations du composant IS sont notifiées à l'application *Rc_display* par le biais de *TrueCall*. En effet la classe *RcReceiveInfo* a besoin d'être informée des informations qui changent à tout moment. Lorsqu'il y a des nouvelles informations dont l'application *Rc_displayfinal* a fait une souscription auprès de IS pour la réception, l'IS invoque automatiquement une methode de la classe *TrueCall* pour lui signifier l'arrivées de nouvelles informations. La classe *TrueCall* notifie ces informations à la classe *RcReceiveInfo* pour la mise à jour des informations de *Rc_displayfinal*. La classe *TrueCall* fait en quelque sort office d'interface entre la classe *RcReceiveInfo* et le composant IS pour la reception des informations.
- ◆ Entre la classe *RcCommands* et la classe *RcReceiveInfo*, il existe une relation d'association de nom de "update information" de cardinalité exactement égale à 1. C'est grâce à cette association que les informations de la classe *RcCommands* qui est une interface graphique contenue dans l'interface graphique de la classe mère *Rc_displayfinal* sont mises à jour.

- ◆ Entre la classe *RcStatusDisplay* et la classe *RcReceiveInfo*, il existe une relation d'association de nom de "update information" de cardinalité égale exactement à 1. Grâce à cette association, les informations de la classe *RcStatusDisplay* qui est une interface graphique contenue dans l'interface graphique de la classe mère sont mises à jour.
- ◆ Il existe le même type de relation d'association "update information" entre la classe *RcErrorCommands* et la classe *RcReceiveInfo*.

Diagramme 4.6 : Le modèle d'objets du «GUI_RC Interface».

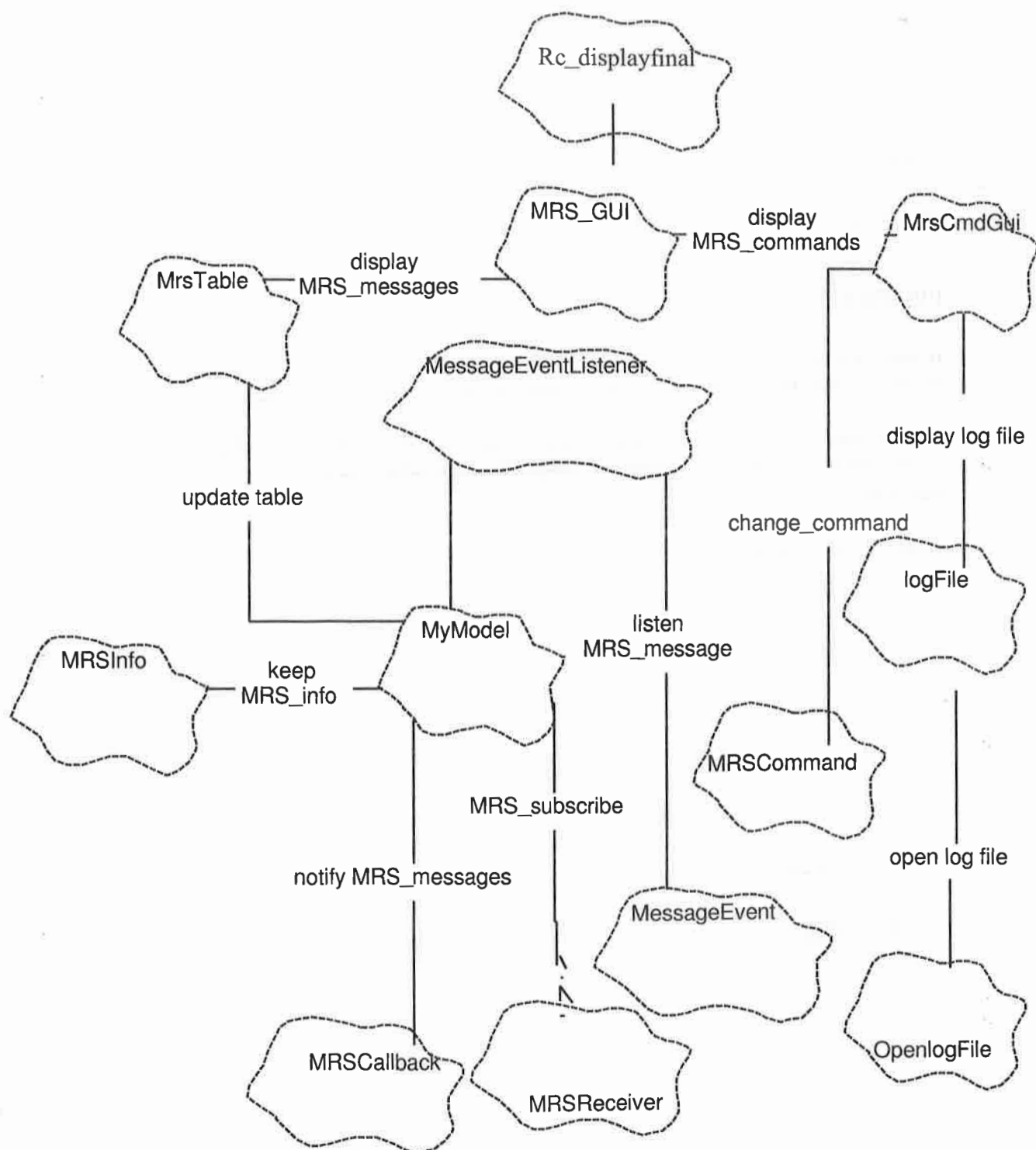
(Context)

ControlDisplayCategorie



4.2.7 Modèle d'objets du composant GUI_MRS Interface

On montre uniquement pour ce composant le diagramme du modèle d'objet sans entrer dans les détails d'analyse. Les attributs et les methodes membres ne sont pas indiqués dans les objets. Seuls les noms des objets le sont.



MRS_diagram on 10/22/1999

Diagramme 4.7 Modèle d'objets du composant GUI_MRS Interface.

4.2.8 Contexte de GUI intégrant les composants GUI_RC Interface et GUI_MRS Interface

Au démarrage du GUI, elle va lire la configuration du système dans la base de données du DAQ et les différents paramètres. Une fois que le chargement est terminé, le GUI vous permet d'afficher la liste des différentes partitions du système DAQ et d'en choisir une comme la partition de travail. Cela permet aux différents utilisateurs du GUI de travailler chacun dans sa partition sans aucune interférence sur les ressources disponibles dans le DAQ.

Lorsque l'utilisateur choisit une partition de travail, le GUI fait une souscription auprès de l'IS, et cela à travers un objet publié par l'IS pour la réception des informations et en suite charge depuis la base de données du RC les contrôleurs disponibles dans la partition. Ainsi l'utilisateur peut gérer les différents contrôleurs chargés grâce aux différentes interfaces graphiques que constitue le GUI.

Lorsque l'utilisateur choisit un contrôleur, le GUI interroge la base de données du RC pour obtenir l'état courant de ce contrôleur et par la suite interagit avec les composants le RC et l'IS afin d'obtenir une instance «le handler» de chacun des objets publiés par le RC et l'IS.

Il était prévu qu'au cours d'une session d'acquisition de données puisse changer de partition de travail qui assure une souplesse suffisante à l'interface, mais malheureusement au moment de l'implémentation de ce projet certains paramètres nécessaires à l'implémentation de cette option étaient en cours de définition et n'étaient pas encore disponibles. Mais plus tard on s'est aperçu que cette option n'était pas compatible avec l'objectif visé par le système de partitionnement. Donc la version du GUI de ce projet n'implémente pas cette option.

Nous donnerons un exemple d'exécution du GUI après l'exposé de chacun des composants. La figure 4.2 montre le contexte du GUI avec ses composants.

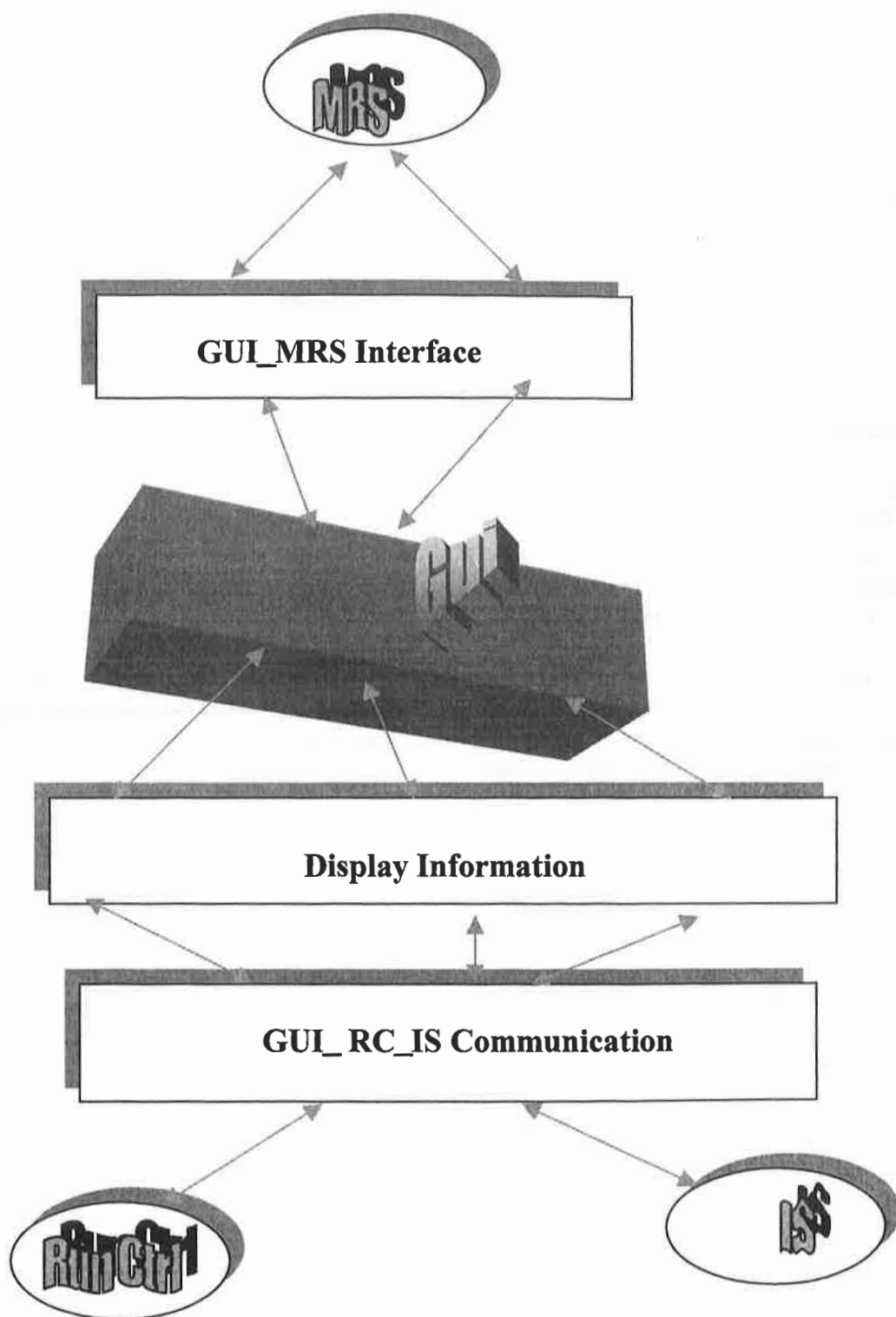


Figure 4.2 : Le contexte du GUI intégrant les composants « Display Information », « GUI_RC_IS Communication » et « GUI_MRS Interface ».

4.3 Le composant «GUI_RC Interface»

4.3.1 L'Interface Graphique Utilisateur «*RCCommands*»

Cette interface permet d'envoyer les commandes aux contrôleurs pour gérer et contrôler les activités du DAQ. Elle est l'interface graphique qui représente la machine à état fini CHSM exposé dans le chapitre 4. Tous les états et les transitions de cette machine sont représentés sur cette interface. Les transitions sont représentées sous forme de commandes exécutables que l'opérateur peut envoyer pour donner des ordres de changement d'état aux contrôleurs. A chaque commande envoyée correspond une action.

Dans la machine CHSM, lorsque les expressions conditionnelles sont évaluées à «true», alors la transition correspondante est franchie et la machine rentre dans l'état correspondant. Le fait qu'une expression conditionnelle soit vérifiée est mise en œuvre dans l'interface «RC Commandes» par l'activation des commandes et les états correspondants et la désactivation de toutes les autres commandes et états. Ainsi le franchissement d'une transition de la machine à état fini CHSM correspond à un envoi de la commande au système RC. Un clique sur une commande équivaut à l'envoi du code numérique de cette commande au composant Run Control qui examine le code et à partir de là, retourne au composant IS les résultats adéquats des actions relatives à ce code, qui à son tour met à jour l'interface *RCCommands*.

L'envoi de toutes les commandes à Run Control et toutes les réceptions d'informations en provenance d'IS par cette interface graphique utilisent les services de communications offerts par le module «GUI_RC_IS Communication». La figure 4.3 montre l'interface graphique *ReCommands* isolée des autres modules.

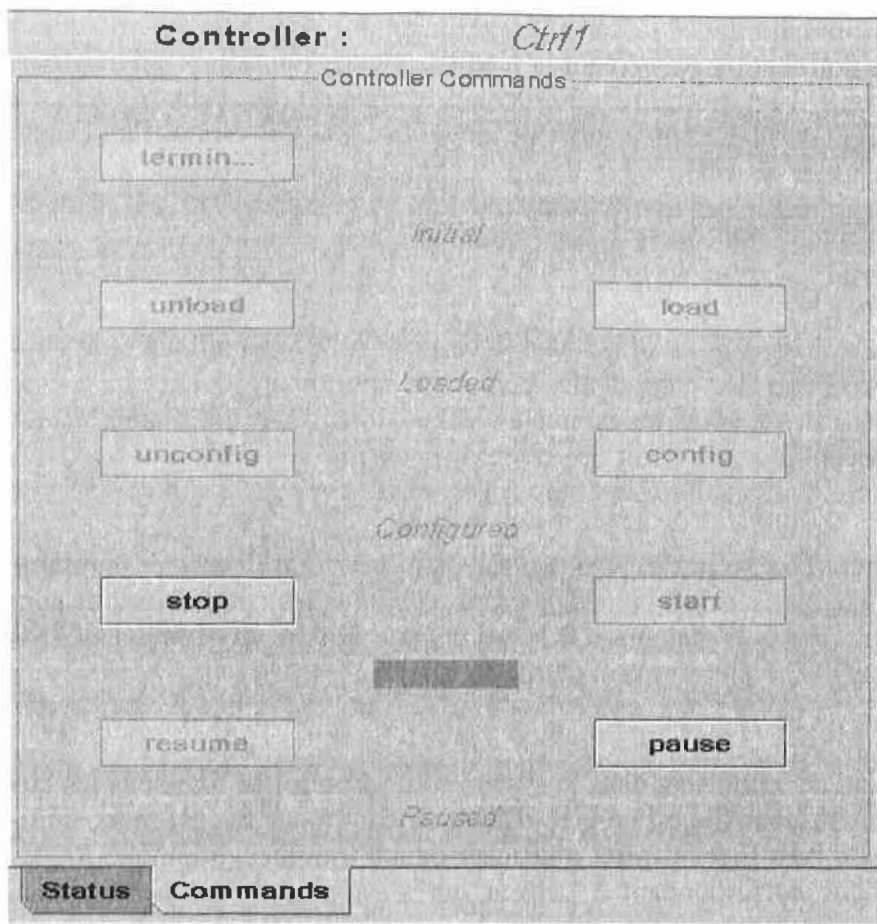


Figure 4.3 L'interface graphique *RcCommands* isolée des autres modules.

4.3.2 L'Interface Graphique des statuts des contrôleurs «*RcStatusDisplay*»

L'ensemble unique de fonctionnalités offertes par cette interface graphique se situe dans l'affichage de certaines informations globales concernant les éléments des dispositifs électroniques sous le contrôle du contrôleur courant. Elle permet de visualiser graphiquement pendant tout le long du processus d'acquisition de données les informations sur le statut du contrôleur courant notamment les informations qui concernent :

- ◆ la Présence;
- ◆ le Lock Status;
- ◆ le DAQ Fault;
- ◆ Last Command;
- ◆ Last Error;
- ◆ Last Sender.

Le *RcStatutsDispay* informe à tout moment l'utilisateur des erreurs dans l'hérarchie du Run Control.

Présence

Dans un système d'acquisition de données tel que DAQ, il est important de pouvoir visualiser dans une interface graphique l'état de la présence des contrôleurs dans l'arborescence hiérarchique. Après l'initialisation, le contrôleur courant lit à partir de la base de données dynami-

ques du Run Control la configuration pour se construire en mémoire la liste de ses enfants. Il reçoit du composant IS une notification concernant la présence de chacun de ses enfants indiquant si l'enfant est en vie ou s'il est mort. Lorsqu'un contrôleur est notifié «Alive» cela veut dire que le processus représentant le contrôleur est en exécution. Lorsqu'il est notifié «Dead» cela veut dire que le processus est absent.

Ainsi à chaque envoi de commande par le «RcCommanades» au composant RC, le «RcStatusDisplay» récupère et affiche le statuts de la présence du contrôleur.

Lock Status

Le Lock status affiche le statut courant de «LockMgr» (voir §4.2.4.7) qui indique si le contrôleur est libre ou non de recevoir une commande. Lorsque le contrôleur démarre lui-même ou traite une commande, l'état de «LockMgr» est mis à «Busy», lorsqu'il est libre ou qu'il est prêt à traiter de nouveau, des commandes l'état de «LockMgr» est mis à «Free».

DAQ Fault

Le DAQ Fault affiche le statut de l'hierarchie des contrôleurs. Si le contrôleur a un problème avec un de ses fils, le DAQ Fault est mis à la valeur «Bad» et si tous les contrôleurs fils sont en vie alors le DAQ Fault est à la valeur «OK». A chaque envoi de commande par l'opérateur au contrôleur, le DAQ Fault est mis à «OK» ou «Bad».

Last Error

Rappelons que les erreurs qui se produisent dans le système Run Control se font sous les circonstances suivantes : Un contrôleur fils est en état «Bad», un contrôleur fils est mort, un timeout s'est produit, etc... Le Last Error permet d'afficher dans l'interface graphique *RcStatusDisplay* le code numérique correspondant à l'erreur qui s'est produit qui permet ainsi l'identification et la récupération d'erreurs.

Last Sender

Cette partie de l'interface est très importante dans le sens qu'elle permet l'affichage de l'identité de l'instance de l'application cliente qui a envoyé la dernière commande au RC. En effet comme plusieurs instances d'interfaces graphiques et autres applications peuvent s'exécuter en même temps et envoyer chacune quand elle le désire une commande au RC, alors il devient nécessaire pour les autres d'identifier celle qui vient d'envoyer la dernière commande. Ainsi à chaque réception d'une commande par RC, l'instance de l'application qui a envoyé la commande est affichée dans Last Sender.

Notons que le *RcStatusDisplay* ne fait aucun envoi de commandes ou d'informations à RC, mais reçoit seulement des nouvelles informations. Le *RcStatusDisplay* utilise les services de l'interface de réception *RcReceiveCommands* du module «GUI_RC_IS Communication» pour toutes les mises à jour. La figure 4.4 ci-après montre l'interface *RcStatusDisplay* isolée contenant certaines informations sur le statut du système.

Presence :	Alive
Lock Status :	Free
DAQ fault: :	OK
Last Command:	resu me
Last Error:	301
Last Sender:	RcSendCommands@18eb606

Figure 4.4 Le *RcStatusDisplay* isolé contenant quelques informations sur le statut du système.

4.3.3 L'interface Graphique «*RcErrorCommands*»

Dans le chapitre précédent nous avons vu la puissance du «Membership» (voir §4.2.7) dans la gestion de l'hierarchie du RC. C'est pourquoi, il était nécessaire de construire une interface graphique pouvant offrir les fonctionnalités de «membership». Ainsi le *RcErrorCommands* offre à l'opérateur la possibilité d'exécution des commandes qui permettent de gérer le «membership» :

- «In» : Lorsque cette commande est exécutée par l'opérateur le code est envoyé à RC et le contrôleur participe aux contrôles d'acquisition des données du groupe de contrôleurs qui constituent le «membership».
- «Out» : L'envoi de la commande «Out» par l'opérateur à RC provoque l'exclusion du contrôleur des contrôles d'acquisition des données de son «membership».

L'interface offre également les commandes «Reset» et «kill». «Reset» permet quand elle est reçue par le RC de réinitialiser tous les contrôleurs, cela est nécessaire lorsqu'une erreur se produit dans l'hierarchie des contrôleurs tandis que la commande «kill» permet de tuer tous les processus contrôleurs.

Le fait de regrouper les fonctionnalités de «membership» et les deux commandes «Reset», «kill» fut un simple choix de conception et d'implémentation. On ne peut pas avoir lorsqu'il y'a plusieurs interfaces qui s'exécutent en même temps, un contrôleur marqué en «In» dans une interface et le même contrôleur marqué en «Out» dans les autres interfaces. Un contrôleur quand il est inclus dans le «membership» («In») quelque part alors il l'est dans toutes les autres interfaces en exécutions. Quand il est exclus «Out» du «membership» quelque part alors il l'est dans toutes les autres interfaces en exécution.

Pour communiquer avec RC, le «*RcErrorCommands*» doit utiliser les services de l'interface d'envoi de commande «*RcSendCommands*» du module «*GUI_RC_IS Communication*».

Pour la réception des notifications des informations en provenance du composant IS, il utilise les services du «RcReceiveCommands» du module «GUI_RC_IS Communication». La figure 4.5 montre le «RcErrorCommands» isolé des autres modules.

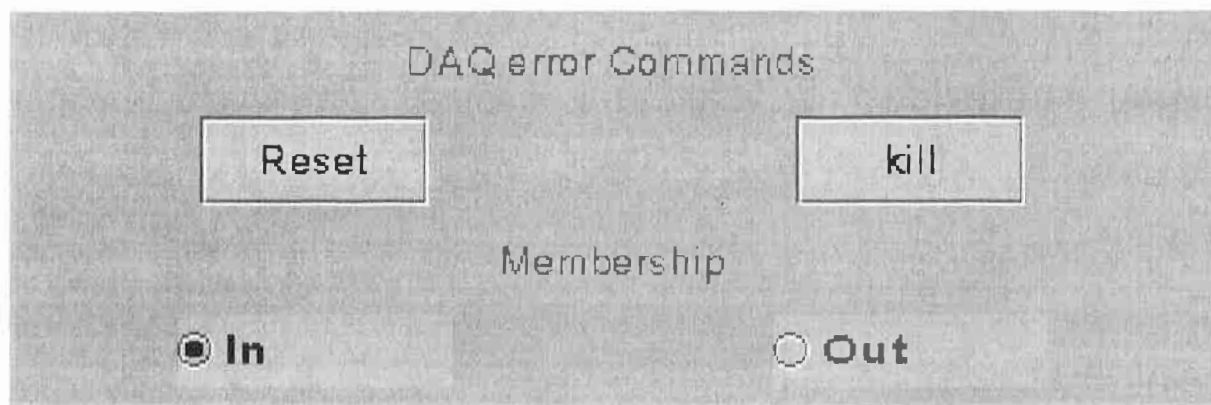


Figure 4.5 Le *RcErrorCommands* isolé des autres modules.

4.3.4 Le sous-composant «GUI_RC_IS Communication»

Ce composant est la passerelle de communication entre le composant «GUI_RC Interface» et le composant RC d'une part et entre le même «GUI_RC Interface» et le composant IS d'autre part. Il utilise la communication du standard CORBA. Les objets utilisés dans la communication sont des objets générés du compilateur IDL (Interface Definition Language) du ILU qui est une implémentation du standard CORBA.

Le module «GUI_RC IS Communication» est structuré en deux sous-modules logiques l'un offrant les fonctionnalités liées au transfert d'informations et l'autre la réception des informations. Le premier sous-module est constitué d'une interface d'envoi de commandes et d'informations appelée *RcSendCommand* qui prend en charge tous les aspects du transfert des commandes du «GUI_RC Interface» vers le composant RC. Les fonctionnalités les plus importantes offertes par cette partie sont l'envoi des commandes et la recherche de l'objet *rc_handle* publié par le RC. La méthode *send()* de ce module utilise la méthode *send_command()* de l'interface «commander» du RC qui est implémentée en C++. Pour pouvoir utiliser ces objets du RC afin de permettre la communication avec le RC, on utilise l'ILU pour générer du code Java. Voici un exemple de spécification IDL l'objet *commander* du RC:

```
module rc{

typedef long Command;

interface commander{
    oneway void send_command(in string name, in Command cmd);
};
};
```

Le deuxième sous-module est constitué d'une interface de réception d'informations appelée *RcReceiveCommands*, elle prend en charge tous les soucis de réception des nouvelles informations en provenance d'IS. Les fonctionnalités les plus importantes offertes par cette interface sont la réception des informations et la recherche de l'objet *is_handle* publié par IS. Elle s'occupe également de la souscription auprès de l'IS et donne la possibilité de connaître les derniers états des contrôleurs de la dernière session d'acquisitions de données. Des méthodes de ce sous-module utilise également des méthodes et objets de l'IS pour communiquer.

Comme les modules de l'IS sont également implémentés en C++, il a été nécessaire d'utiliser également le compilateur de l'ILU pour générer du code Java. Voici un exemple de spécification IDL de certains objets de l'IS :

```
module is{

enum Reason{ Created, Updated, Deleted };

struct Item{
    string      name;
    string      type;
    string      data;
    string      time;
    is::Reason   reason;
};

typedef sequence<Item>    list;

interface callback{
    oneway void  notify(in is::Item item);
};

interface factory{

    exception    NotFound{};
    exception    NotCompatible{};

    long         create_info      (in string name, in string type, in string data);
    void         delete_info      (in string name) raises (NotFound);
    void         get_info_list    (in string rexp, out list l);
    long         subscribe        (in string name, in callback cb);
    long         unsubscribe      (in string name, in long id);
    long         query_subscribe  (in string rexp, in callback cb);
    long         query_unsubscribe(in long id);
    void         set_value        (in string name, in string type, in string value)
raises (NotFound,NotCompatible);
    void         get_value        (in string name, out string type, out string value,
out string time) raises (NotFound);
    void         get_type        (in string name, out string type) raises
(NotFound);
};
};
```

4.4 Le composant «GUI_MRS Interface»

4.4.1 L'interface graphique des informations formatées du MRS «MrsTable»

Nous savons que le composant MRS transporte et distribue les messages d'erreurs. Au niveau du destinataire ces messages reçus doivent être affichés pour les analyser. Mais le format sous lequel ces messages sont reçus n'est pas lisible par un utilisateur non expert. L'objectif de ce module est de trouver un moyen de synthétiser les messages d'erreurs envoyés par le MRS pour ne retenir dans le message que les informations pertinentes et de trouver un moyen

[illegible]

4.4.2 L'Interface graphique d'envoi de commandes MRS «MrsCmdGui»

La deuxième partie implémente les commandes qui permettent de changer la souscription auprès du MRS. En effet le MRS n'envoie que des types de messages auxquels on a fait une souscription. On peut à tout moment changer cette souscription.

66

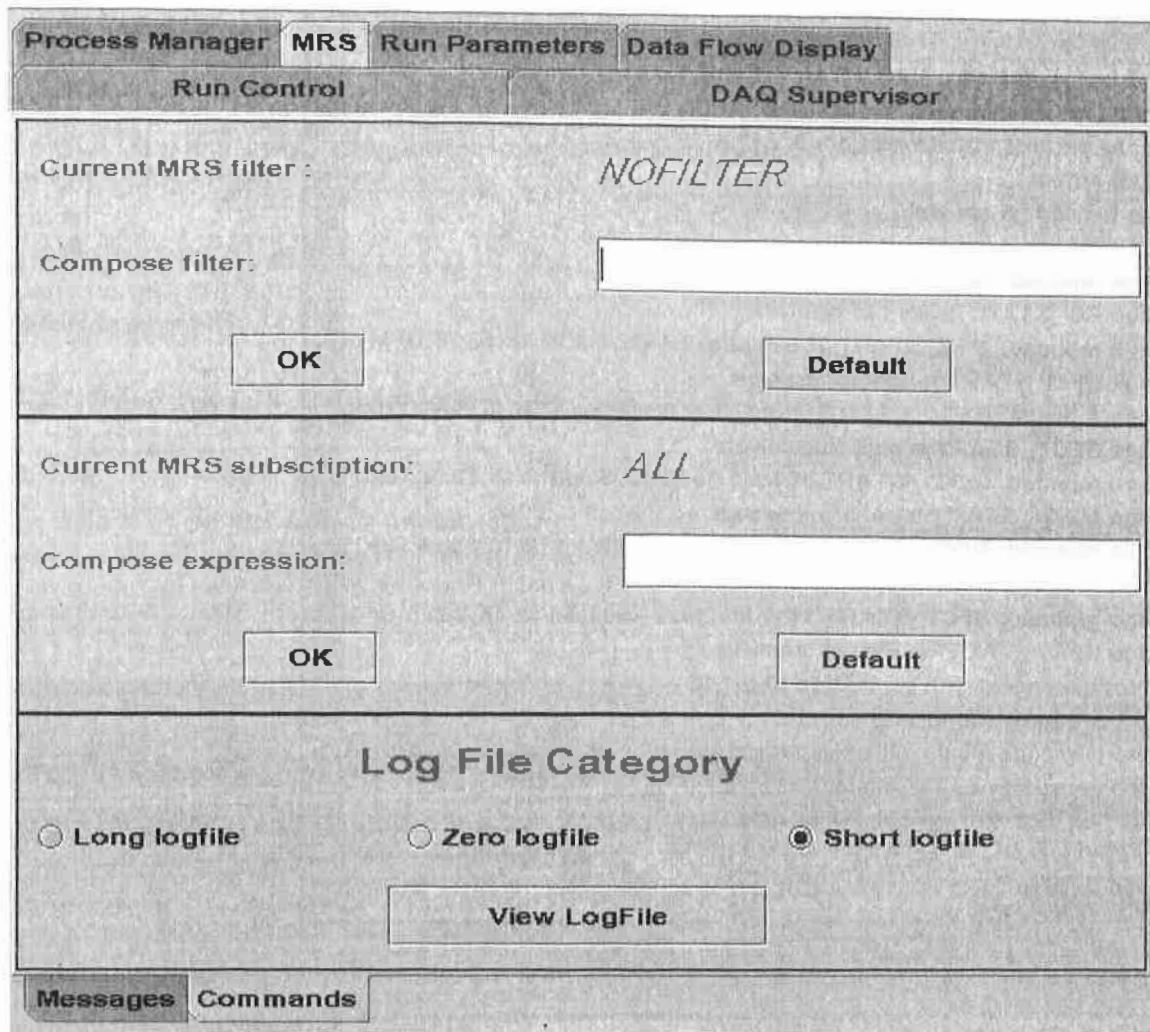


Figure 4.7 L'interface *MrsCmdGui* isolée des autres modules.

4.4.3 L'interface «Mrs logfile Display»

Ce sous-composant communique uniquement avec le «MrsCmdGui». Lorsqu'un opérateur a besoin de visualiser le logfile, il envoie la requête par la commande «viewLogFile» à ce sous-composant qui, une fois reçu la requête, va chercher le fichier logfile créé dans la partition du travail de l'opérateur par le composant MRS, l'ouvre en lecture, lit tout le contenu du fichier comme une longue chaîne de caractères, et crée une fenêtre avec un *textArea* pour afficher cette chaîne de caractères dans le *textArea* de la fenêtre. Ce sous-composant implémente les services de la localisation du fichier logfile, de sa lecture et s'occupe de la partie graphique de l'affichage. La figure 4.8 montre le contenu d'un fichier logfile avec le format «Short logfile».

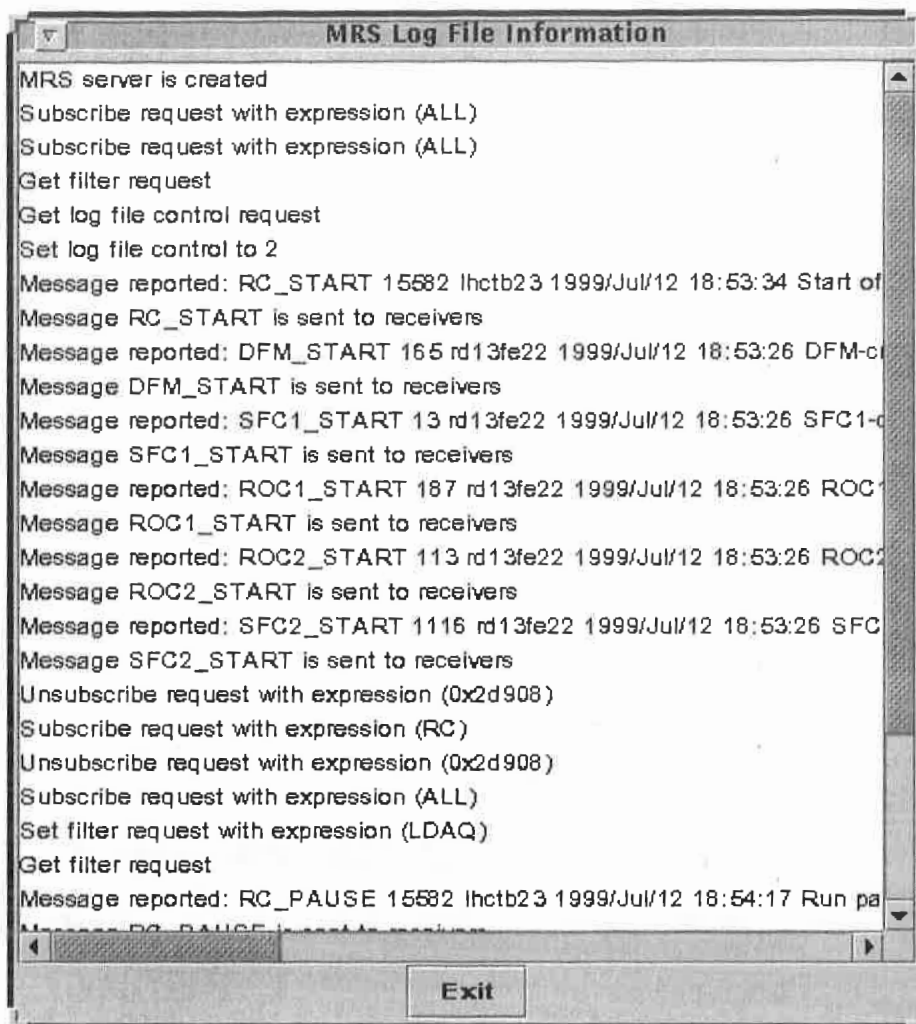


Figure4.8 Le « Mrs logfile Display » contenant le fichier logfile avec le format Short logfile.

4.5 Conclusion

Ce chapitre élabore la manière dont on représente dans une interface graphique utilisateur les interactions entre l'IS, le MRS, le RC et les programmes externes ou auxiliaires au système DAQ. Deux composants logiciels ont été construits à cet effet : le «GUI_RC Interface» et le «GUI_MRS Interface».

Le «GUI_RC Interface», constitué de deux sous-composants le «Display Information» et le «GUI_RC_IS Communication», prend en charge les interactions entre l'IS, le RC et les programmes externes. Le sous-composant «Display Information», constitué de différentes interfaces graphiques du composant «GUI_RC Interface», s'occupe de la partie graphique tandis que le sous-composant «GUI_RC_IS Communication» s'occupe de la communication entre ces interfaces graphiques et l'IS.

Composé du *MrsTable*, du *MrsCmdGui* et d'un sous-composant « Mrs logfile Display», le composant «GUI_MRS Interface» permet la souscription auprès du MRS, l'affichage et le filtrage des messages d'erreurs. Il fournit également les fonctionnalités de contrôle du logfile du MRS.

5. L'intégration de GUI

Le GUI (version finale) intègre tous les modules aussi bien les modules élaborés dans ce projet de diplôme que les autres modules élaborés par d'autres personnes des autres groupes de travail.

Dans ce qui suit nous montrons les résultats de l'intégration des modules élaborés dans le cadre de ce projet.

5.1 Résultat de l'intégration du sous-composant «*RcCommandes*» dans GUI

La figure 5.1 montre le sous-composant *RcCommandes* intégré à l'interface graphique GUI qui gère l'ensemble du système d'acquisition de données DAQ/EF :

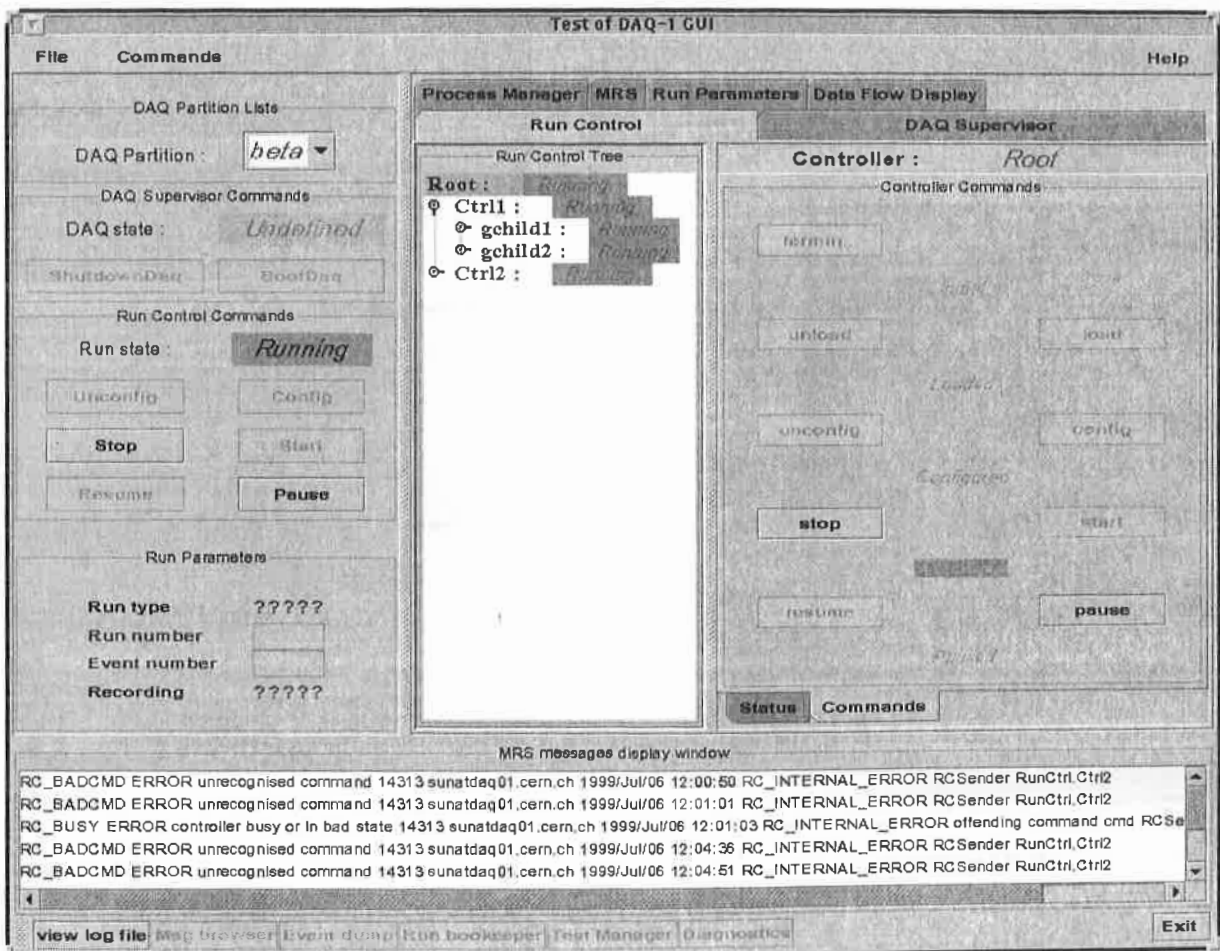


Figure 5.1 Le GUI intégrant le *RcCommandes*.

5.2 Résultat de l'intégration des sous-composant « RcStatusDisplay » et « RcErrorCommands » dans le GUI

La figure 5.2 montre les deux modules *RcStatusDisplay* et *RcError commands* intégrés au reste du système par le biais du GUI.

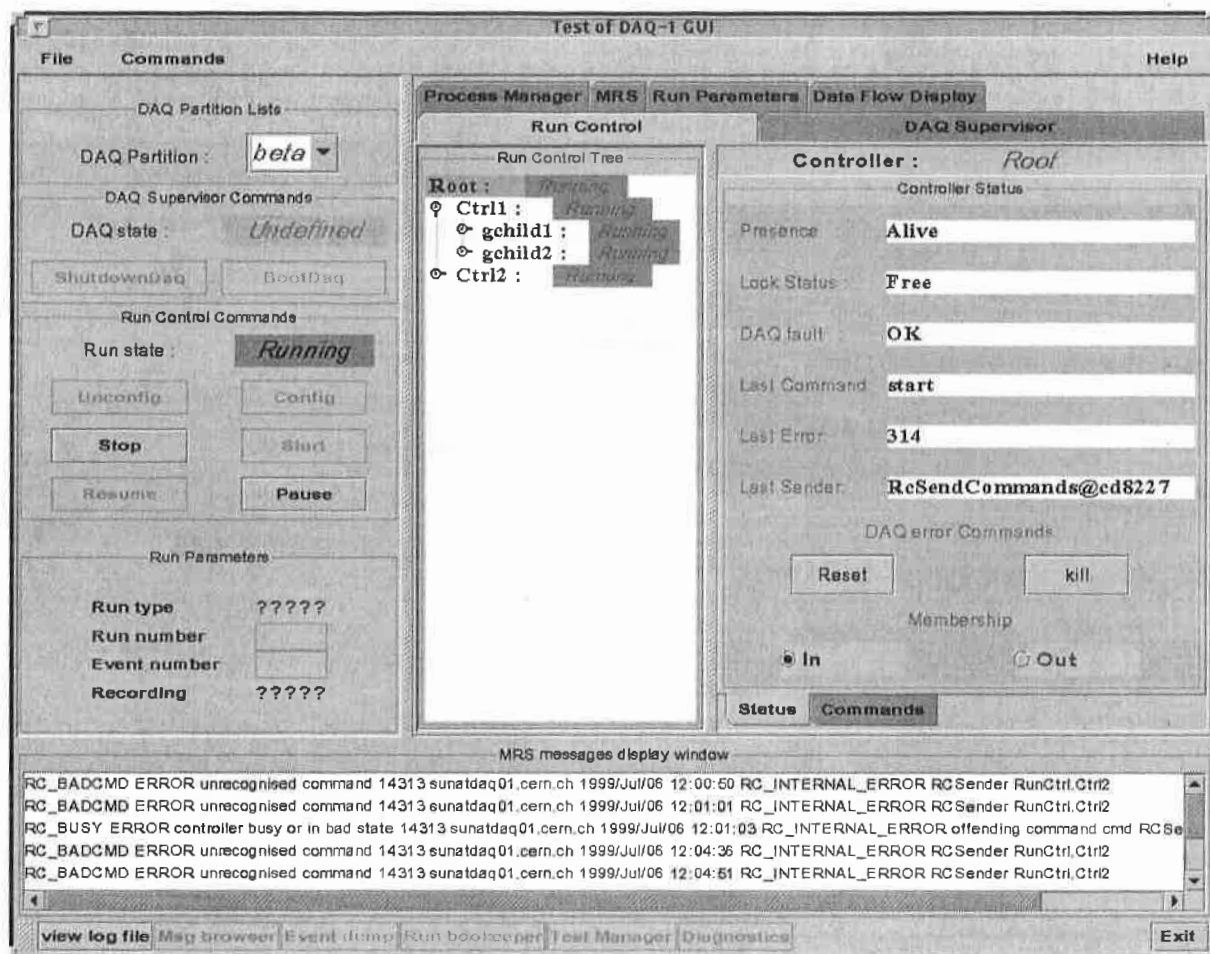


Figure 5.2 Le *RcStatusDisplay* et le *RcErrorCommands* intégrés au GUI .

5.3 Résultat de l'intégration du sous-composant «MrsTable» dans le GUI

La figure 5.3 montre MrsTable intégré et s'exécutant dans l'ensemble du système GUI.

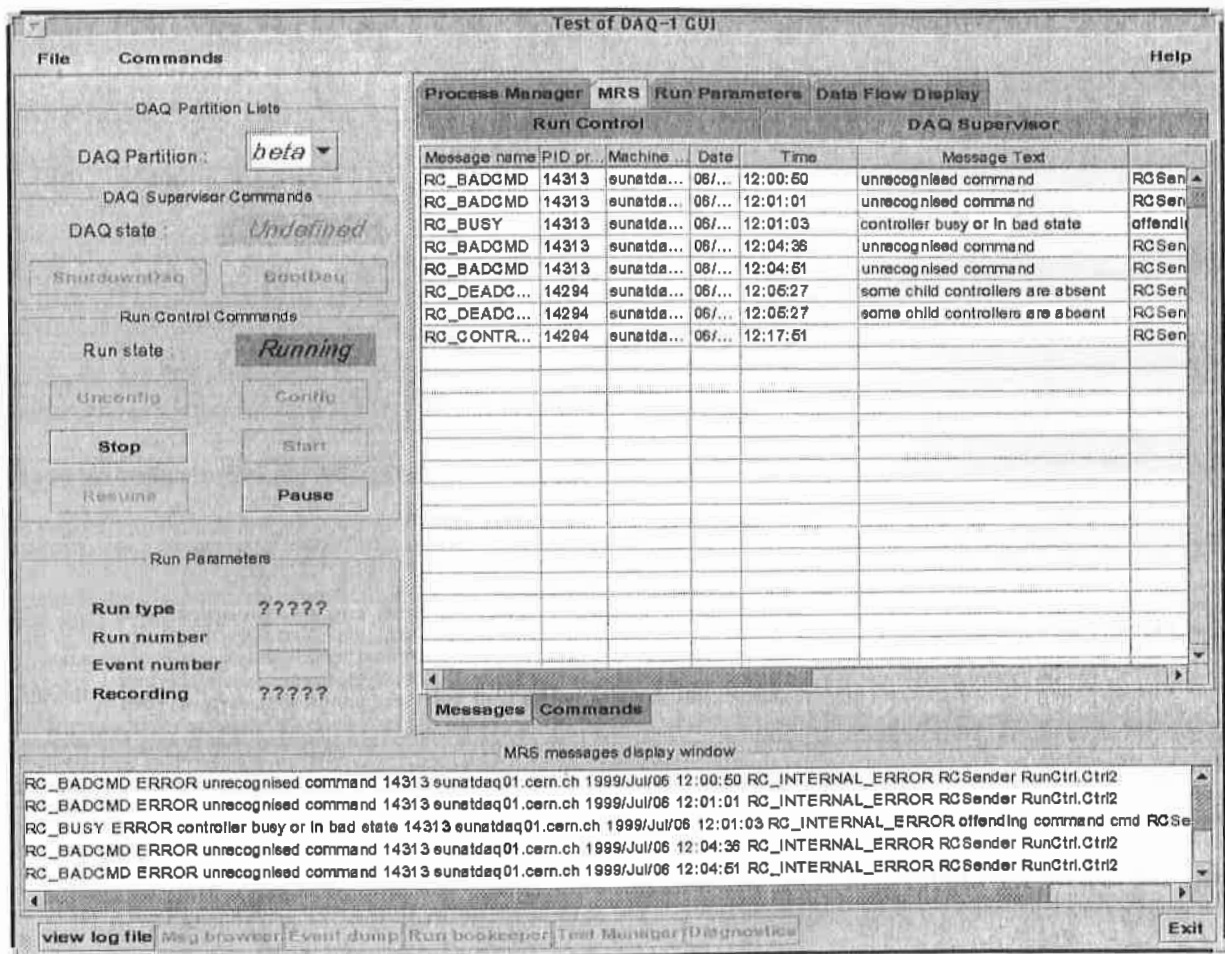


Figure 5.3 Le Mrs Table intégré et s'exécutant dans l'ensemble du système GUI.

5.4 Résultat de l'intégration du sous-composant « MrsCmdGui » dans GUI

La figure 5.4 montre l'exécution du module *MrsCmdGui* dans l'ensemble du système après l'avoir intégré au GUI.

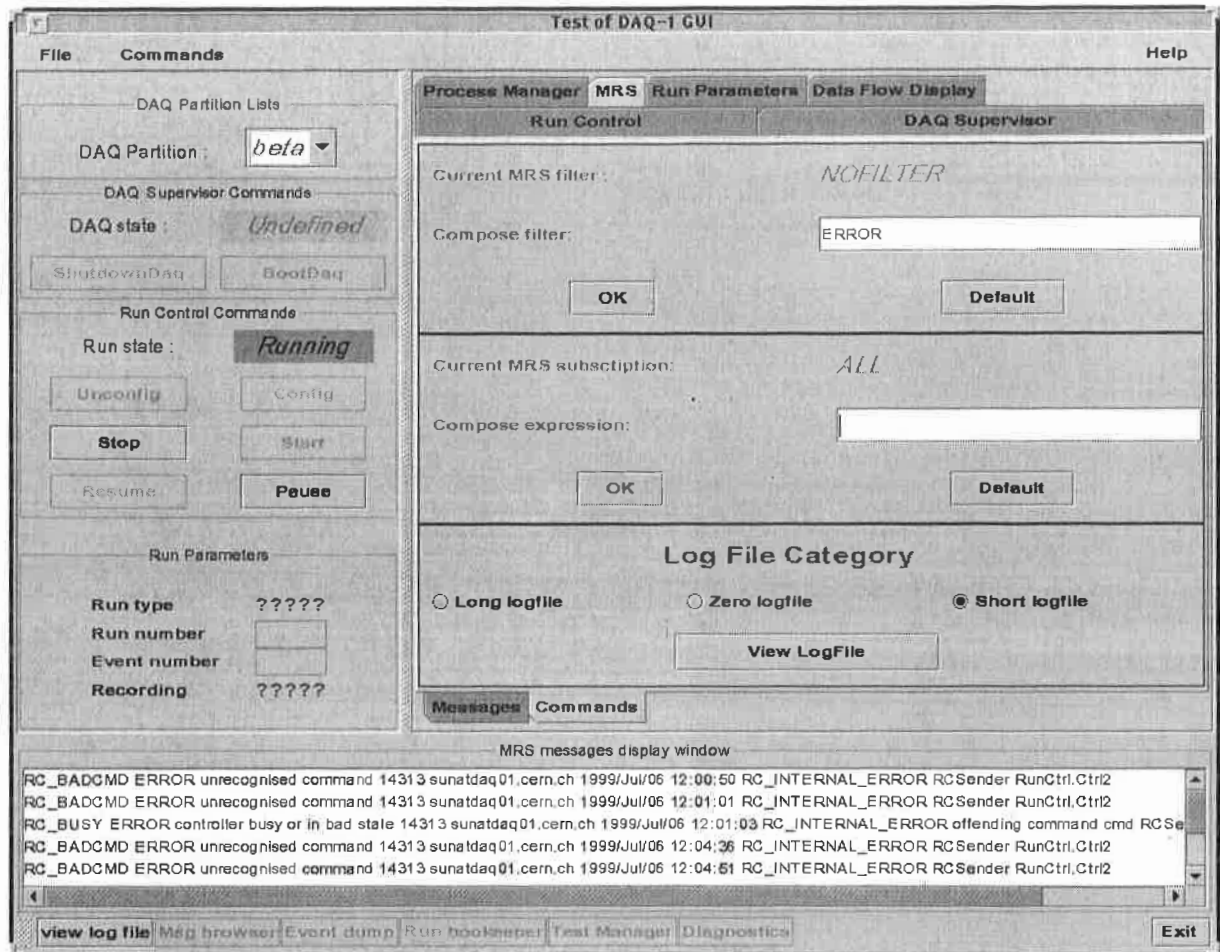


Figure 5.4 Le GUI contenant le *MrsCmdGui*.

5.5 Pourquoi JAVA ?

Un système moderne d'acquisition de données est un système complexe et large de taille qui exige une interface graphique utilisateur sophistiquée. X11 et Motif sont des outils technologique interactifs largement dominants sous UNIX, mais l'avènement de Windows NT nous a forcé à reconsidérer ce choix. JAVA est un langage orienté objet simple, multi-thread, et garantissant l'indépendance de plat-forme. Il fut procédé à une évaluation des software sur le marché. Le but de cette évaluation a été de comprendre si JAVA pourrait être utilisé pour implémenter les composants de l'interface graphique du DAQ. Une application de démonstration a été développée au CERN qui comprend le GUI standard, une structure client/serveur, et utilise des méthodes natives, des Widgets spécifiques pour l'affichage et des «remote objects». La performance a été comparée avec un alternatif X11. La démonstration contenait trois parties essentielles : les serveurs, les simulateurs et les applets. Les serveurs réalisent le

lien avec les «remote objects», les simulateurs créent la simulation des données et mettent à jour les «remote objects». (pour simuler le DAQ), les applets fournissent les données de simulations aux «remote objects» ou récupèrent les données des «remote objects» pour des fins d'affichage. La totalité de l'application a été écrite en JAVA (JDK1.0). Les communications ont été réalisées par IDL (Apha2). Les serveurs et les simulateurs s'exécutent sur SUN (solaris2.5) et les applets peuvent s'exécuter sur toutes les machines ayant un browser compatible JAVA. Les conclusions de cette démonstration ont montrées qu'il était nettement avantageux d'utiliser JAVA.

Nous avons utilisé le CASE tool VISAJ pour la conception des parties graphiques du GUI. VISAJ est un outil graphique totalement interactif, écrit complètement en JAVA et qui génère du code pure JAVA.

5.6 Conclusion

Après le teste de chaque sous-composant élaboré dans le chapitre précédent dans une version teste du GUI, ils ont été intégrés dans la version finale qui n'a pas posé de problèmes majeures au niveau des fonctionnalités de chaque composant. Le délai limite qui est l'an 2000, pour mettre en exploitation le GUI a pu être respecté. Cependant les testes d'évaluations continueront au CERN jusqu'à l'an 2000 pour diagnostiquer d'éventuelles anomalies. Le contrôle d'identification pendant l'accès à certaines options du GUI et la correction d'éventuels anomalies détectés avant l'an 2000 seront l'objet d'implémentation dans une nouvelle version du GUI.

1. Les différents niveaux des triggers

1.1 Le niveau-1 de Triggers (LVL1)

Au niveau-1 (LVL1), des processeurs dédiés travaillent sur des données de granularité fine des sous-ensembles de détecteurs (Calorimètre et muon trigger chambre). Le LVL1 accepte les données à un taux LHC bunch-crossing de 40MHz (à toutes les 25ns). La latence (temps pris pour former et distribuer la décision de LVL1 triggers) est environ 2 microsecondes. Pendant ce moment les données des détecteurs sont dans la mémoire de pipeline en attendant une décision. Le taux maximal accepté est de 75kHz et est déterminé par la capacité du sous-système de stockage Readout. Atlas exige de porter cette capacité à 100kHz et le taux maximal de sortie attendu pourrait ainsi atteindre 100kHz. A une certaine LHC luminosité chaque bunch-crossing produit une moyenne d'environ 23 proton-proton collisions qui résultent en une sortie de données totales d'au moins 100 Gbytes/s dans le système d'acquisition de données.

Après l'acceptation LVL1 triggers, les données sont enlevées des détecteurs et envoyées pour être stockées dans le système Readout pendant le processus de LVL2 triggers et pendant le temps de collection pour la construction d'événements.

Le LVL2 doit réduire ce taux à environ 1kHz. Son architecture est basée sur l'utilisation des régions intéressantes (ROIs).

Le LVL1 est utilisé pour identifier, pour chaque événement, les régions de détecteurs qui contiennent des caractéristiques intéressantes telles que: une High- P_T de clusters électromagnétiques, de jets et de muons. Le LVL2 doit alors accéder et travailler sur une petite portion (fraction) de la totalité des données des détecteurs qui est un avantage en terme de la puissance de calcul et de la capacité de mouvement des données. Toutes ces précisions complètes sur les données des détecteurs aussi bien des calorimètres que les muons sont utilisées par le LVL2. Des données de sous-ensembles de détecteurs sont combinées pour fournir une meilleure identification de particules et une haute précision de mesure qui sont possible dans les LVL1 triggers. Cela est nécessaire si le taux de réduction exigé par LVL2 doit être atteint. Le temps moyen de décision est estimé à environ 10ms.

Lorsqu'un événement est accepté par les LVL2 triggers, toutes les données pour cet événement sont envoyées à un processeur d'EF via l'EB (constructeur d'événement). Une reconstruction complète d'événement est possible à ce troisième niveau et le temps de décision au tour de 1 seconde est exigé.

Le LVL2 relie les informations de LVL1 pour mener le processus et les mouvements de sélection de donnée. Dans le mécanisme de ROI, le LVL1 identifie pour LVL2 les régions spatiales des détecteur qui contiennent des objets candidats. Ces objets peuvent être des muons, des électrons/protons, taus/hadrons ou jets. Le LVL1 triggers fournit des informations sur la position de chacun de ces objets et son taux de porté d'énergie. Il fournit également à LVL2 des renseignements sur les propriétés globales des événements: le scalaire total E_T et le vecteur E_T^{miss} et il spécifie quelles signatures commandent l'événement sélectionné par LVL1 trigger. La figure 1.1 montre quelles données et signaux sont passés de LVL1 à LVL2.

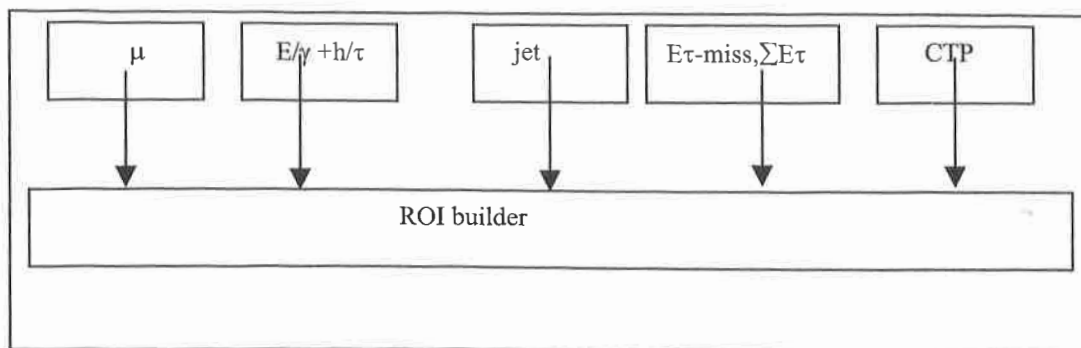


Figure 1.1 Les signaux qui sont passés de LVL1 à LVL2.

L'information est fournie d'un certain nombre de sources dans LVL1 : trois dans le calorimètre, et un de chaque dans le muon et le processeur central de trigger. Dans chaque cas les données sont accompagnées par des événements et des identificateurs bunch-crossing.

La ROI builder du système LVL2 trigger utilisera l'information fournie pour distinguer entre la ROI «primaire», ces objets qui ont pu contribués directement à la sélection des événements au niveau LVL1 et la ROI «secondaire», les objets marqués par LVL1 mais n'ayant pas joués directement de rôles dans la sélection des événements. Une Low- P_T de jet est un exemple typiquement d'une ROI secondaire.

1.2 Niveau-2 trigger

Dans cette section, on introduit la description du second niveau du système triggers de l'ATLAS.

Le niveau-2 (LVL2) agit sur les événements aux taux retenus par le niveau-1 qui est estimé à 75kHz (incluant les facteurs de sécurités) et qui peut être étendu jusqu'environ 100kHz. Son rôle est de réduire ce taux à un niveau supportable par le système EB de l'ordre de quelques kilohertz. Dans le concept courant de la notion de granularité grossière, la précision complète devraient être disponibles. Le troisième niveau appelé le système EF opère après le système EB et réduira le taux des données jusqu'à ce qu'il soit égal à une capacité de stockage permanent.

La division des taches entre LVL2 et le système EF est déterminée de sorte que cela contribue à l'optimisation globale de système triggers (coût, performance).

Le mode normale d'opération du système LVL2 trigger pour High- P_T trigger est basé sur l'utilisation des ROIs, définie par le système LVL1. Une région intéressante est signalée par LVL1 pour les candidats muons, électrons/photons, taus/hadrons et les jets. Les informations sur ROIs, en particulier leur position η et Φ et le critère de sélection LVL1, sont disponibles pour LVL2; Basé sur cette information, LVL2 décide quelles données de chaque sous-détecteur sont associées à chacune des ROIs et seulement ces données sont transférées aux processeurs LVL2.

Cette sélection de données de LVL2 réduit la charge sur le système LVL2 et permet une simplification des algorithmes de LVL2.

Le mode d'opération proposé de LVL2 trigger est une séquence d'étapes: extraction, object building (construction d'objets) et les sélections triggers.

Dans les extractions, les données de chaque ROI pour chacun des détecteurs sont recueillies ensemble et traitées pour fournir des quantités de données appropriées (Par exemple pour le calorimètre, ce processus prendrait des cellules d'informations et produirait des paramètres en clusters; pour les détecteurs de traces, les informations basiques affectées sont converties en des paramètres de traces ou des segments de traces).

La construction d'objet prend ces caractéristiques pour chaque région venant de tous ces détecteurs considérés et retourne les particules de ces paramètres ou des grains de ces paramètres, et si possible, les «types» de ces paramètres. Si les données sont conformes à plus qu'un «type» de particule, le sous-processus de traitement fait des testes avec chaque «type» d'objet possible.

Dans la phase de la sélection trigger des «types», tous les objets trouvés dans les événements sont combinés et comparés avec la topologie (types des particules, des impulsions transverses, inclusive/missing mass, etc.) d'un menu de sélection physique. Des flags sont mis pour chaque élément du menu pour lesquels une correspondance est trouvée.

1.2.1 Sélection des événements (Event selection)

La tâche du système trigger est la sélection des événements suivant le menu de trigger. Les éléments dans le menu de sélection physique s'expriment en termes de 7 objets triggers de base: muons, électrons, photons, hadrons chargés, taus, jets et E_T^{miss} (la discrimination entre électrons et photons est possible due aux tracks d'informations valables dans LVL2).

Chaque élément dans le menu de trigger définit un ensemble de conditions de trigger incluant le nombre d'objets exigés de chaque type et les exigences des propriétés de ces objets. (e.g. le seuil de la porté de l'énergie P_T , le critère d'isolation, etc.). Les caractéristiques des conditions triggers s'appliquent à plusieurs objets triggers qui peuvent aussi être définies, tel que la réduction sur des masses effectives. Pour chaque objet trigger, il existe réellement plus qu'un seuil d'énergie d'impulsion transverse.

Le seuil minimal réel correspond au seuil du trigger pour les objets participants à un trigger multi-objets. Le seuil maximal réel correspond à un single objet de trigger non isolé. La confirmation des objets trigger peut être une étape décisive dans la sélection. Cependant la non confirmation d'un objet trigger candidat ne permet pas nécessairement la rejection d'un événement.

Le critère de sélection trigger sera considérablement moins desserré que leur utilisation pour l'analyse finale et doit être bien tenue suffisamment flexible pour permettre des sélections efficaces de nouveaux et intéressants phénomènes physiques. Le LVL2 trigger permettra la réduction proportionnelle des éléments du menu de trigger, et une acceptation forcée de certains événements.

1.2.2 Le mode d'opération et de partition

Le système LVL2 supporte plusieurs modes d'opérations. En mode normale, le système est utilisé pour la sélection d'événements et seuls les événements acceptés sont passés au EF. Dans le mode étiquette, ou (tagging mode) le système traite tous les événements mais ne rejette pas d'événement. Tous les événements sont passés au EF.

Pour ces deux modes, le système LVL2 opère sous le contrôle du Back-end. Le mode teste se limite à tester des procédures qui utilisent soit le teste ou soit les événements de données.

Les testes peuvent impliquer plusieurs portions de triggers et de chaînes d'acquisitions de données ou peut être des testes internes, standalone au système LVL2. Pour les testes n'utilisant pas le Back-end DAQ, des contrôles locaux et système DAQ local seront utilisés.

Les systèmes de sous-détecteurs doivent être capables d'opérer indépendamment durant le développement, la calibration et le teste, donc il doit être possible de partitionner le système LVL2. Pour opérer indépendamment, chacune des partitions a ses propres contrôles locaux et ses propres systèmes DAQ locaux.

1.2.3 Description fonctionnelle des composants du niveau-2

Le système LVL2 peut être vu en terme de quatre blocs fonctionnels principaux : le complexe ROB, le superviseur, le réseau et l'environnement de traitement.

Cette section ne prend pas en compte certaines particularités de l'architecture ou de l'implémentation, mais essaie d'identifier et décrire les différentes caractéristiques des fonctions logiques des quatre blocs.

L'analyse détaillée et la manière dont ces fonctions sont regroupées et liées entre elles sort du cadre d'études de ce projet.

1.2.3.1 Le Complexe ROB

Pour les événements acceptés par LVL1 les données de tous les canaux électroniques du Front-end sont transférées dans le readout buffers (ROB).

Le ROB est un composant fonctionnel standard commun à LVL2 et aux flux de données du DAQ/EF tous les deux. Sa fonction principale est de stocker les données des fragments événements durant la latence du LVL2.

Pour la communication avec le système LVL2, il existe une interface de contrôle et une interface des données. L'interface de contrôle reçoit des requêtes pour des données et des décisions venant du système LVL2. L'interface de données envoie des données demandées du ROB vers le système LVL2. L'interface de données peut avoir des options de pré-traitements de données de façon locale au ROB et des options pour combiner des données de plusieurs ROB (via un bus ou le réseau) si nécessaire pour la transmission.

1.2.3.2 Le Superviseur et Le ROI Builder

Le superviseur et le ROI Builder ont trois fonctions principales. La première est de recevoir au taux d'acceptation LVL1 (jusqu'à 100MHz) les fragments d'informations venant de LVL1 et construit l'enregistrement (record) de ROI pour chaque événement. Les enregistrements ROIs contiennent une liste de ROIs avec leurs «types» et positions η , ϕ . La seconde fonction est d'allouer les processeurs de LVL2 suivant les besoins et de router les enregistrements ROI vers les processeurs adéquats du système LVL2 (les processeurs qui ont été affectés au traitement des données des événements ROI); La troisième fonction est de recevoir les décisions LVL2 et de les envoyer à tous les ROB (c'est-à-dire un Broadcast au complexe ROB).

1.2.3.3 Le Réseau

Ils existent deux réseaux logiques : le réseau de collection de données qui transmet les données ROI aux différents éléments de traitements; le réseau de contrôle qui se soucie de l'acceptation LVL1, des requêtes de ROI de LVL1 et des messages accept/reject de LVL2. Dépendant de l'architecture et de l'implémentation choisie, chacun de ces réseaux pourrait

être implémenté comme plusieurs réseaux physiques ou ils peuvent être combinés dans un seul réseaux pour tout le trafic.

1.2.3.4 L'environnement de traitement

Ils existent trois taches principales qui peuvent être effectuées en utilisant les processeurs élémentaires correspondants aux types principaux des algorithmes utilisés dans les étapes de sélections LVL2 :

- ◆ Extraction des caractéristiques, qui traite les données d'une seule ROI et d'un seul détecteur pour extraire les caractéristiques telles que le calorimètre de clusters d'énergies et des paramètres de traces.
- ◆ La construction d'objets, qui combine les caractéristiques pour un seuil de ROI des différents détecteurs et retourne les propriétés des particules et les «types» possibles.
- ◆ Traitement global, qui combine les objets et les compare aux éléments du menu trigger pour produire l'acceptation ou le rejet de décision LVL2.

Les processeurs peuvent être groupés en clusters ou subfarms. Les subfarms devraient être connectés au réseau via les switches de l'interface Farm (L2-SFI).

1.3 Problèmes et Contraintes

Le problème technique fondamental pour l'implémentation de LVL2 est comment rendre disponibles les données aux processeurs exécutant les algorithmes des triggers et comment contrôler l'utilisation de ces processeurs. La taille globale de chaque événement est ~ 1 Mbyte. Ils sont reçus au taux d'acceptation de LVL1 (environ 100kHz), les données de chaque événement s'étendent sur 1700 ROBs du système, la puissance de traitement (calcul) exigée par les algorithmes LVL2 est de ~ 10 ms.

Plusieurs stratégies ont été proposées qui individuellement ou en combinaison rendent ce problème plus atreillant :

- ◆ Transmettre au système LVL2 seulement les données contenues dans ROIs trouvées par le système. Cela réduit les données à quelques pourcentages du total pour chaque événement mais introduirait les contraintes pour distribuer les données demandées à des ROBs appropriés.
- ◆ Ces caractéristiques de réduction du volume de données qui ont été transmises peuvent être obtenues pour certains sous-détecteurs par le prétraitement des données dans ou auprès des ROBs, par exemple en réformatant les données pour qu'elles soient plus compactes.
- ◆ Utiliser les processeurs en parallèle. Cela peut être fait à plusieurs niveaux : traitements des différents événements; l'algorithme de traitement dans un événement, c'est à dire l'extraction des caractéristiques de différentes ROIs ou différents sous détecteurs, le traitement des données de différents ROBs ou groupe des ROBs;

traitement associé au ramassage des données. Ces différentes formes du traitement parallèle affecteraient le taux et la latence de décision LVL2, mais ils ont aussi un impact sur le nombre total des processeurs et des problèmes d'allocations aussi bien que des contrôles de processeurs.

- ♦ Utiliser les processeurs de très grande vitesse à exécuter les algorithmes d'extractions de caractéristiques. Cela réduit le nombre de processeurs et simplifierait les problèmes de contrôle mais introduirait un nouveau problème de collection de données.
- ♦ Construire le réseau qui relie les ROB's aux processeurs venant d'un nombre de petits réseaux. Cela par ce qu'en générale, il est plus simple de construire plusieurs petits réseaux qu'un unique réseau large; particulièrement si le réseau reste non-bloquant (c'est à dire l'utilisation de certaines paires des ports avec un bandwidth maximal ne réduit pas le bandwidth disponible entre d'autres paires de ports).
- ♦ De réduire le nombre d'événements qui demandent le traitement de plusieurs algorithmes en appliquant la sélection séquentielle. Par exemple, dans la première étape de traitement, d'utiliser seulement les ROIs trigger dans les systèmes calorimètres et muons, et alors procédé à d'autres traitements d'événements si le LVL1 trigger est confirmé. Cela peut mener à une bonne optimisation de bandwidth de données et du nombre total de traitements exigés. L'extension de cela à plusieurs étapes conduirait à plus de flexibilité des triggers, bien qu'il affecterait le degré auquel le traitement parallèle peut être appliqué dans un événement. Le nombre d'étapes croîtrait aussi bien que le nombre de phases de requêtes de données (mais le nombre total de requêtes de données ne changerait pas). Cela a des implications pour le mécanisme utilisé pour transmettre les requêtes aux ROB's et, d'éviter des attentes vides des données (Idle time), qui impose donc un chauvauchement de traitements des différents événements dans des processeurs conduisant à l'accroissement de temps de switch.

Les deux premières de ces stratégies ont été retenues pour toute l'implémentation de LVL2 trigger d'ATLAS, le degré de traitement parallèle et d'autres stratégies continuent à être des axes de recherches. L'importance de chaque stratégie et les options qui permettent de les combiner impliquent des changements des contraintes variées.

Du côté technique, l'amélioration continuelle des composants de haute technologie sur le marché promet des systèmes majeures avec des solutions simples.

L'évolution croissante des processeurs RISC/CISC continue de suivre un accroissement vertigineux qui double approximativement la puissance de calcul tous les 18 mois.

Les processeurs de 300-500 MIPS standards augmenteraient de 2-4 GIPS au début des opérations LHC et peut être à 10GIPS à l'exécution à haute luminosité indiquant que la puissance de calcul pure pour l'extraction des caractéristiques et de traitement global ne posent plus problème (bien que cela demande encore un accroissement de performance correspondante des périphériques d'entrée/sortie (I/O)). Similairement la taille et la puissance des dispositifs électroniques continuent à se réduire et accroître respectivement, et déjà ils ont démontré leurs efficacités.

Les puces mémoires utilisés pour le stockage des données dans les ROB's sont devenus plus larges avec la conséquence qu'une latence LVL2 d'environ 10ms n'est plus un problème majeure.

Le bandwidth exigé pour le trafic total de réseau entier de LVL2 est estimé à être de l'ordre de Gbyte/s. Cet haut débit de bandwidth ne peut pas être basé sur les bus traditionnels de système d'acquisition de données. La plus récente haute technologie des composants réseaux sur le marché promet un niveau de performance exigé, si cette tendance continue à un prix abordable. La construction d'un réseau d'une telle grande performance serait toujours une tâche complexe et pour parvenir, les exigences de bandwidth et de latence, cela demanderait des extrêmes efficacités dans les composants et l'expérience nécessaire. Il est maintenant approprié d'accomplir des évaluations détaillées et des comparaisons de ces technologies.

Le système LVL2 complet serait très complexe et exigerait une maintenance à long terme. Pour réduire le temps moyen de développement et de maintenance, il sera grandement préférable d'utiliser des composants disponibles dans le commerce pour la configuration modulaire lorsque c'est possible.

L'extraction des données des ROB et le mouvement des données entre les processeurs sont des problèmes majeurs. Du côté hardware, les interfaces et les dispositifs I/O entre les différents éléments de grandes vitesses (liens et réseaux) sont généralement des régions critiques à optimiser. Du côté software, les performances des divers dispositifs, et les overheads divers tels que le contexte des switches et les interruptions auront besoin d'être optimisés.

L'extraction et la collection des données ROIs des ROB imposent des contraintes critiques sur la conception du ROB et d'autres composants. Le nombre des ROB qui contribuent à chaque ROI et le taux auquel chaque ROB est accédé sont grandement couplés pour mapper les éléments détecteurs dans les ROB. Les détails de mapping de chaque sous-détecteur dans les ROB pourraient avoir un impact majeur sur la performance du système LVL2.

Le format des fragments d'événements de données peut affecter le volume de données qui sont transférées dans le système LVL2, et/ou la performance des algorithmes d'extraction des caractéristiques. Dans certains cas le reformatage des données sera désirable avant le transfert à travers le réseau. L'optimisation des formats qui viennent des détecteurs pourrait faire une importante économie de bandwidth et minimise la manipulation des données dans le ROB ou dans le système LVL-2, dans d'autres cas l'optimisation pourrait être effectuée par pré-traitement au niveau du ROD, mais clairement, cela influencerait la conception du ROD.

La balance entre la complexité des RODs, les bénéfices des simplifications des extractions des données du ROB et le formatage est seulement partiellement comprise à ce stade et mérite des investigations approfondies. Il est donc important que les formats soient choisis de façon à minimiser les overheads des extractions, des transferts et certains formatages nécessaires des données restreintes exigés par LVL2.

C'est une grande exigence de LVL2 et un problème réel s'il n'était pas satisfait.

Finalement le système LVL2 doit être flexible, qui permet différents scénarios de sélections d'événements et de changement des limites avec le EF comme l'accroissement du mécanisme de luminosité de LHC.

1.4 Les Paramètres d'entrée

Cette section présente les différents paramètres d'entrée pour le système venant de la simulation physique et la spécification des détecteurs électroniques. Certains nombres sont objet d'avantage d'optimisation, mais cet ensemble est un ingrédient de base pour les prochaines étapes de modélisation du LVL2.

1.4.1 Les informations de LVL1

Des exemples de menu de LVL1 et de taux d'entrée pour le processus de sélection LVL2 à haute et faible luminosité sont donnés dans la table 3.1.

LVL1 trigger requirement	Rate in kHz
MU20	4
MU6x2	1
EM30I	22
EM20Ix2	5
J290	0.2
J130x3	0.2
J90x4	0.2
J100+XE100	0.5
T60+XE60	1
MU10+EM15I	0.4
Other triggers	5
Total	40

Table 3.1 Menu de LVL1 et de taux d'entrée pour le processus de sélection LVL2 à haute et faible luminosité.

Un taux moyen de LVL1 trigger de $\sim 40\text{kHz}$ est compatible avec les $75\text{-}100\text{kHz}$ qui est la limite conceptuelle et permet quelques marges de sécurité.

1.4.2 Les informations fournies par LVL1 à LVL2

Les informations qui passent de LVL1 à LVL2 sont résumées ci dessous :

- ◆ High- P_T muons ROIs :
 - La portée P_T (les muons sont classifiés en six portées de l'énergie P_T);
 - L'emplacement du ROI dans le détecteur avec typiquement une résolution η - ϕ de 0.1×0.1 ;
 - Flags (qui indiquent la possibilité de chauvauchement de chambre, overflows dans la résolution des candidats muons proche de la plus grande P_T muon d'un secteur).
- ◆ High- P_T électron/photon ROIs :
 - La portée P_T (les clusters électromagnétiques isolés sont classifiés en huit 8 portées P_T);
 - Emplacement du ROI dans le détecteur avec typiquement une résolution η - ϕ de 0.2×0.2 .
- ◆ High- P_T tau/hadron ROIs :
 - La portée P_T (les clusters hadronics sont classifiés en huit 8 portées P_T);
 - L'emplacement ou la location du ROI dans le détecteur avec typiquement une résolution η - ϕ de 0.4×0.4 .

- ◆ L'information scalaire et E_T^{miss} :
 - La valeur scalaire de E_T ;
 - La valeur E_x et E_y (du vecteur E_T^{miss}).
- ◆ Le «type» d'information de trigger :
 - Multiplicité du seuil de la portée P_T pour chaque type d'objets;
 - Résultat de comparaison de seuil de scalaire et missing E_T ;
 - D'autres entrées trigger (c'est à dire la calibration spéciales des triggers);
 - Les informations sur les menus de LVL1 pour chaque critère de sélection qui est rencontré (avant et après l'application de pré-calibration).

Dans chaque cas des informations d'entête seront fournies donnant l'identificateur d'événements et les numéros d'identificateurs des «bunch-crossing» aussi bien que quelques bytes pour les flags d'erreurs et le formatage. Pour les informations sur l'implémentation du LVL1 trigger incluant ses interfaces à LVL2, cela sort du cadre de ce projet.

1.5 Processus de sélection et stratégie

Le LVL2 trigger utilise une granularité grossière des données venant des muons, des calorimètres et d'autres pour créer les objets trigger (viz particules ou des candidats jets certifiés par une séquence d'algorithmes). Le code LVL2 trigger est optimisé pour l'exécution rapide utilisant un minimum d'informations des ROIs définie par LVL12 trigger.

Plusieurs triggers physiques seront implémentés tels que : les triggers inclusifs, des triggers sur les quantités globales et combinaison de signature, les β -triggers. Les triggers pour les mesures de luminosité et le contrôle des faisceaux, la calibration et l'alignement seront aussi implémentés.

Nous assumons que le LVL2 peut réduire le taux de trigger par un facteur d'environ 100. Cela est réalisé en taillant le seuil des triggers inclusifs et en accomplissant quelques traitements complexes qui sont possibles dans LVL1. En particulier LVL2 est le premier endroit où les données des détecteurs de traces sont disponibles.

Les données ROIs sont traitées par les algorithmes d'extraction de «type» pour extraire des quantités qui sont utilisées dans les décisions globales. Les méthodes de sélections sont sans itérations et plus simplifiées que dans les codes offlines. Les algorithmes ne dépendent pas de la résolution finale des détecteurs puisque l'ensemble complet de calibration de données ne pourrait pas être disponible.

Les muons identifiés au LVL1 seront confirmés au LVL2 dans le spectromètre de muons qui utilisent les deux chambres de données de trigger et de précision. (rejets en fond arrière et réduction du seuil). Les détecteurs d'intérieurs pourraient être utilisés pour une confirmation additionnelle et l'isolation des candidats muons serait contrôlés avec les données du calorimètre.

Pour les électrons/photons identifiés au LVL1, la plupart d'analyses redéfinies au LVL2, qui utilisent des précisions complètes, des granularités grossières des données de calorimètre seront utilisées pour rejeter en arrière plan les jets et diminuer le seuil. D'avantages de raffinements des électrons candidats peuvent être accomplis en comparant les données de calorimètre avec les paramètres de détecteurs d'intérieurs de traces et en utilisant la signature de transition des radiations.

Taus (decaying to hadrons) et les hadrons isolés identifiés au LVL1 seront sujets d'analyses redéfinies au LVL2 utilisant la précision complète, une granularité grossière des données, de calorimètre à rejeter des jets en arrière plan pour diminuer le seuil.

Les seuils pour les jets identifiés à LVL1 seront diminués en faisant plus d'analyses redéfinies au LVL2 qui utilisent des précisions complètes de données de calorimètre. Cela cependant ne donne pas un grand nombre de jets que l'on peut rejeter en arrière plan indiquant que les jets sont des processus dominants. Les informations de détecteurs d'intérieurs pourraient être utilisées pour identifier les β -jets au LVL2 en utilisant des paramètres d'algorithmes.

La recalculation complète au LVL2 pour la signature de E_T^{miss} de LVL1 ne peut pas être obtenue à un taux élevé, mais des corrections pour les high- P_T muons, les mauvaises mesures de jets etc., pourraient être appliquées. Il pourrait cependant être possible de faire des recalculations complètes pour une petite fraction d'événements.

Nous donnons dans la section qui suit une description brève des extractions de «type» dans les différents détecteurs.

1.5.1 Extraction de «type» Muons

L'algorithme muons identifie les pistes de muons dans les régions ROIs sélectionnées par le muon LVL1 trigger et accomplit un calcul précis de leur emplacement. Une granularité grossière est utilisée pour permettre la diminution du seuil de P_T , qui réduit ainsi le taux de LVL1 muons par un facteur de deux.

Dans chaque ROI sélectionnée par LVL1 trigger ($\Delta\eta \times \Delta\phi \sim 0.1 \times 0.1$), l'algorithme accomplit deux étapes : la reconnaissance des pistes de muons dans les chambres de précision et des impulsions transverses convenables.

1.5.2 Extraction de «type» Calorimètre

L'algorithme d'extraction de type calorimètre cherche les clusters dans les données de calorimètre électromagnétique et hadron en restreignant les données aux ROIs.

Pour chaque cluster trouvé, le centre de gravité en η et ϕ , les énergies des cascades et les autres quantités décrivant les propriétés transversales et longitudinales sont calculés incluant les paramètres isolés. Les quantités dans les premiers compartiments électromagnétiques et calorimètres sont aussi calculées. Dans ce qui suit, nous résumons les algorithmes d'extractions de «type» électromagnétiques (EM), les signatures de tau/hadron, et les jets incluant les E_T^{miss} et la somme totale des énergies transverses $\sum E_T$.

Pour ces objets triggers, le LVL1 trigger peut être utilisé jusqu'aux huit seuils, où les seuils des clusters EM consistent à un triplet de seuils : E_T de cluster, E_T de EM d'isolations et E_T d'hadron d'isolation. Pour chaque objet LVL1 trigger accepté, le seuil et les informations du ROI sont transmis à LVL2. La position de ROI est fournie avec une résolution comparable à la taille des cellules LVL1 trigger, c'est à dire pour le cluster EM, la précision est : $d\eta \times d\phi = 0.1 \times 0.1$. L'algorithme LVL1 travaille sur une matrice de $\eta \times \phi$ qui stocke l'énergie par colonnes de triggers et des deux EM et hadron calorimètre ensemble. Ces matrices sont disponibles dans LVL2 pour des contrôles et vérifications.

LVL2 redéfinit les informations de LVL1 en utilisant la granularité grossière et la précision complètes. Les données sont collectées des ROBs des régions $d\eta \times d\phi$ d'une taille déterminée par la précision de la position de ROI et le «type» d'algorithme à exécuter. Typiquement, la décision LVL1 est vérifiée en recalculant et en développant les positions des clusters en η et ϕ . Toute analyse future est alors accomplie dans la fenêtre centrée à cette position.

1.5.3 L'extraction de «type» e/γ

Les événements sélectionnés par LVL1 trigger de e/γ sont dominés par l'interaction d'échange de charge des pions isolés et de cascades hadroniques étroites. Ils peuvent être rejetés par l'analyse des formes de cascades d'énergies dans le calorimètre EM et hadronic, et par des analyses détaillées du développement simple de cascades d'énergies d'EM avec sa granularité fine en η . L'algorithme d'identification de e/γ utilise différentes tailles de fenêtre de 3×3 à 7×9 des cellules standards où une cellule standard a la taille de $d\eta \times d\phi \sim 0.025 \times 0.0025$.

La fenêtre utilisée pour collectionner l'énergie est allongée suffisamment grande pour accepter assez de γ s qui ont été convertis (conversion des électrons en ϕ , dû aux champs électromagnétiques).

A peu près 30% de γ s sont convertis avant le calorimètre, 75% de ces conversions se produisent dans les détecteurs d'intérieur. Des algorithmes ont été élaborés pour détecter des conversions, mais ils sont considérés comme trop compliqués pour être appropriés à LVL2.

La stratégie est donc d'optimiser les critères d'identifications γ tel que γ efficace soit le même pour tous les photons convertis ou non convertis.

1.5.4 L'extraction de «type» Tau/hadron

L'identification de tau relie la sélection de jets étroits isolés, dans les détecteurs de traces à quelques traces. La forme de cascade et l'isolation sont calculées pour les calorimètre EM et hadronic séparément.

L'isolation des clusters est exigée des deux côtés EM et hadronic. Elle est déterminée par la fraction d'énergies déposées dans le cœur de cluster, comparée à l'énergie dans une large fenêtre positionnée au centre du cluster.

1.5.5 L'extraction de «type» jet

L'algorithme LVL2 de jet a pour tâche d'améliorer l'énergie et la mesure de position des jets candidats retrouvés par LVL1. L'amélioration est accomplie en améliorant la calibration d'énergie des seuils nets. Utilisant un cône d'algorithmes sur les informations dans la fenêtre de ROI, cela améliore la reconstruction de la position; la résolution de LVL1 est de 0.2×0.2 dans $d\eta \times d\phi$.

Un important problème est au tour du transfert des données des ROBs vers le système LVL2 associé à l'analyse de jet : beaucoup de ROBs, EM et hadronic ont besoin d'obtenir toutes les énergies dans un ROI de jet (typiquement $d\eta \times d\phi = 0.8 \times 0.8$).

Le bandwidth nécessaire peut être réduit par une pré-sommation des cellules d'énergies dans une matrice en colonnes de triggers ($d\eta \times d\phi = 0.1 \times 0.1$), mais le nombre de transferts de données reste élevé. Nous attendons la sélectivité améliorée au LVL2 pour importer la charge additionnelle pour certains canaux. Une optimisation prudente des seuils sera nécessaire, particu-

lièrement pour la deuxième signature. LVL2 pourrait aussi sélectionner certaines classes de jets pour une analyse détaillée.

1.5.6 Calcul de E_T^{miss} et $\sum E_T$

Le LVL2 trigger reçoit de LVL1 les valeurs scalaires des énergies totales E_T et les composants du vecteur E_T^{miss} : E_x et E_y . Ceux ci sont obtenus en sommant tous les calorimètres d'Atlas ($|\eta| < 4.9$).

Il n'est pas prévu de recalculer ces quantités (sauf si possible pour une petite fraction d'événements à la fin de la procédure de sélection LVL2 en assurant la sélection séquentielle). Cependant des corrections peuvent être appliquées pour le High- P_T muons (qui ne sont pas inclus dans le calcul effectué par LVL1) et pour les mauvaises mesures de High- P_T jets pour qui l'information de ROI sera disponible.

LVL1 implémente un trigger sur $\sum E_T$ qui pour beaucoup de processus physiques peut être utilisé au lieu des valeurs scalaires des énergies totales E_T . Cette quantité peut être recalculée au LVL2 qui utilise l'information de jet améliorée.

1.5.7 Extraction de «type» des détecteur d'intérieurs (TRT, SCT et Pixel)

Une recherche pour les traces (track) dans TRT et les couches de précisions des détecteurs d'intérieurs est utilisée pour compléter l'information du système de calorimètres et de muons. Cela, ensemble a un raffinement des informations des détecteurs utilisés au LVL1, permettant une large réduction à un taux auquel les événements ont été passés dans EF.

La stratégie des triggers des détecteurs d'intérieurs (ID) est de deux catégories :

- ◆ High- P_T trigger, qui est une recherche pour une High- P_T piste (track), ($P_T > \sim 5\text{GeV}$) dans une région limitée à ROI indiquée par LVL1;
- ◆ Low- P_T trigger, qui est une recherche non guidée dans le TRT pour traces avec $P_T > 0.5\text{GeV}$ qui sont extrapolées dans les couches de précisions lorsque c'est nécessaire.

1.6 Conclusion

Dans le système DAQ ils existent trois niveaux de triggers : le LVL1, le LVL2 et l'EF.

Le LVL1 est utilisé pour identifier, pour chaque événement, les régions des détecteurs qui contiennent des caractéristiques intéressantes de données. Il accepte des données à un taux de 40MHz à tous les 25 ns.

Le LVL2 doit agir sur des événements aux taux retenus par le LVL1 qui est de l'ordre de 75kHz à 100kHz. Le LVL2 doit réduire ce taux à un niveau supportable par le système EB d'environ de quelques kHz.

Le troisième niveau des triggers appelé EF opère après le EB et doit réduire le taux des données jusqu'à atteindre une capacité de stockage permanent.

Bibliographie

ATLAS. Technical Progress Report. DAQ, EF, LVL2 and DCS, 30 Jun 1998

Design of the Message Reporting System for the Atlas DAQ prototype-1, 30 January 1997.
Reference : <http://atddoc.cern.ch/Atlas/Notes/032/Note032-1.html>.

ATLAS-1 Process Manager design 26/06/97. Reference:
<http://atddoc.cern.ch/Atlas/Notes/028/Note028-1.html>.

Information Service for Atlas DAQ Prototype-1, 3 February 1997. Reference :
<http://atddoc.cern.ch/Atlas/Notes/031/Note031-1.html>.

ATALAS-1 Run Control System design, 26/06/1997. Reference :
<http://atddoc.cern.ch/Atlas/Notes/029/Note029-1.html>.

Integration with ILU. Reference :
<http://atddoc.cern.ch/Notes/008/Note008-41.html>.

Multiple servers architecture using ILU. Reference :
<http://atddoc.cern.ch/Atlas/Notes/031/Note031-9.html>.