Dissertação de
Mestrado

Processamento em Tempo Real no Experimento MINERνA

Guillermo Arturo Fiorentini Aguirre

Centro Brasileiro de Pesquisas Físicas

Rio de Janeiro, Março 2009

Livros Grátis

http://www.livrosgratis.com.br

Milhares de livros grátis para download.

Agradecimentos

A todas as pessoas que tornaram possível a realização deste trabalho. Ao meu orientador, Dr. Hélio da Motta, pelo constante apoio, atenção, dedicação e amizade, pelos muitos conselhos e ensinamentos recebidos, pela enorme paciência nas intermináveis horas de enriquecedouras discussões "com café e bolo".

Ao Dr. Jorge Morfin por todo o apoio durante minha visita ao FERMILAB.

A todos os pesquisadores do CBPF, em especial ao Dr. Gilvan Alves e Dr. Mário Vaz.

À toda colaboração do experimento MINERνA, em especial ao Dr. Mike Kordosky, ao Dr. Gabriel Perdue, ao Dr. David Schmitz, a Jesse Chvojka, a Jorge Castorena, a Kenyi Hurtado, a Carlos Lopez e a Jaewon Park pelo interesse e disponibilidade para me ajudar em meu trabalho.

Aos colegas e amigos do CBPF, Dr. Cesar Castromonte, José Palomino e David Martinez.

À CAPES pela bolsa concedida durante o Mestrado.

A todo o pessoal administrativo do CBPF, em especial ao Ricardo e a Myriam.

À minha familia por todo o apoio que me deu em todos os momentos.

Resumo

O experimento MINERνA dispõe de um detector de neutrinos altamente segmentado e de grande precisão e registrará uma alta estatística de eventos (mais de 13 milhões em 4 anos de tomada de dados). O MINERνA utiliza a linha de neutrinos muônicos NuMI do FERMILAB. O detector permitirá o estudo de interações neutrino-nucleon em grande detalhe. Em adição, o detector do MINERνA possui um alvo com diferentes materiais, permitindo, pela primeira vez, o estudo de efeitos nucleares em interações de neutrinos. Apresentamos aqui trabalhos realizados junto ao grupo de software do MINERνA que resultaram em: a) desenvolvimento do *software* de processamento e monitoramento em tempo real do MINERνA ; b)instalação de protótipo do detector e tomada de dados com múons cósmicos;

Abstract

MINERνA experiment has a highly segmented and high precision neutrino detector able to record events with high statistic (over 13 millions in a four year run). MINERνA uses FERMILAB NuMI beamline. The detector will allow a detailed study of neutrino-nucleon interactions. Moreover, the detector has a target with different materials allowing, for the first time, the study of nuclear effects in neutrino interactions. We present here the work done with the MINERνA software group that has resulted in: a)development of the MINERνA processing software and online monitoring; b)comissioning of the tracking prototype and data taking with cosmic muons;

Colaboração MINERνA

I. Albayrak[7], L. Aliaga[16], M. Andrews[5], S. Avvakumov[17], B. Baldin[5], J. L. Bazo[16], D. B. Beringer[22] G. Blazey[13], A. Bodek[17], D. Boehnlein[5], S. Boyd[15], R. Bradford[17], W. K. Brooks[10], A. Bruell[10], D. Buchholtz[14], H. Budd[17], A. Butkevich[8], D. Casper[3], C. Castromonte[2], D. Cherdack[20], M. E. Christy[7], J. Chvojka[17], M. A. C. Cummings[13], D. S. Damiani[22], I. Danko[15], M. Day[17] D. Drakoulakos[1], S. Dytman[15], R. Ent[10], G. A. Fiorentini[2], J. Felix[6], R. Flight[17], A. Gago[16], H. Gallagher[20], D. Gaskell[10], R. Gilman[18], C. Gingu[5], C. Glasshausser[18], R. Gran[12], N. Grossman[5], B. Gobbi[14] D. A. Harris[5], J. Hobbs[14] X. Jiang[18], A. Jeffers[14] T. Kafka[20], V. Kuznetsov[14] C. E. Keppel[7], J. Kilmer[5], S. M. S. Kim[15], S. Kopp[19], M. A. Kordosky[22] M. Kostin[5], S. Kulagin[8], G. Kumbartzki[18], H. Lee[17] A. G. Leister[22] L. Loiacano[19], A. Loveridge[14], S. Manly[17], E. Maher[11] W. A. Mann[20], K. McFarland[17], A. McGowan[17] D. Meekins[10], W. Melnitchouk[10], A. Mislevic[17] G. Moreno[6], J. G. Morfin[5], H. da Motta[2], D. Naples[15], J. K. Nelson[22], G. Niculescu[9], I. Niculescu[9], R. Ochoa[21], W. Oliver[20], J. Olsen[5], J. L. Palomino[2], V. Paolone[15], J. Park[17], L. Patrick[14] E. Paschos[4], G. Perdue[17] O. Pereyra[21], A. Pla-Dalmau[5], M. Proga[19], R. Ransome[18], M. Reyes[6], P. Rubinov[5], V. Rykalin[13], C. Rude[12], W. Sakumoto[17], P. Shanahan[5], H. Schellman[14], R. M. Schneider[22], E. Schulte[18], C. Simon[3], J. Solano[21], P. Spentzouris[5], P. Stamoulis[1], J. Steinman[17], J. Tatar[3], V. Tvaskis[7], G. Tzanakos[1], M. Vaz[2], S. Wood[10], G. Zavala[6], B. Ziemer[3], M. Zois[1]

University of Athens, Athens, Greece[1]

Centro Brasileiro de Pesquisas Fisicas, Rio de Janeiro, Brazil[2]

University of California, Irvine, California[3]

University of Dortmund, Dortmund, Germany[4]

Fermi National Accelerator Laboratory, Batavia, Illinois[5]

Universidad de Guanajuato, Guanajuato, Mexico[6]

Hampton University, Hampton, Virginia[7]

Institute for Nuclear Research, Moscow, Russia[8]

James Madison University, Harrisonburg, Virginia[9]

Jefferson Lab, Newport News, Virginia[10]

Massachusetts College of Liberal Arts, North Adams, Massachusetts[11]

University of Minnesota Duluth, Duluth, Minnesota[12]

Northern Illinois University, DeKalb, Illinois[13]

Northwestern University, Evanston, IL[14]

University of Pittsburgh, Pittsburgh, Pennsylvania[15]

Pontificia Universidad Catolica del Peru, Lima, Peru[16]

University of Rochester, Rochester, New York[17]

Rutgers University, New Brunswick, New Jersey[18]

University of Texas-Austin, Texas[19]

Tufts University, Medford, Massachusetts[20]

Universidad Nacional de Ingenieria. Lima, Peru[21]

The College of William and Mary College, Williamsburg, Virginia[22]

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A física de neutrinos é um dos tópicos mais excitantes em física de partículas. Atualmente há evidências de que os neutrinos possuem massa não nula [1] e podem mudar de sabor[2]. A mudança de tipo de neutrino (conhecida como oscilação de neutrinos) foi observada em experimentos com neutrinos solares, neutrinos atmosféricos e em experimento com aceleradores.

Esta tese é sobre meu trabalho realizado no experimento MINERνA, que é um experimento montado na linha de feixe NuMI do FERMILAB empregando um feixe intenso de neutrinos. Atuamos no grupo de software do experimento. Nesta tese apresentamos trabalhos em *software* e *hardware* com vistas a implementação do monitoramento *online* do detector e comissionamento das fotomultiplicadoras.

O Capítulo 2 introduz aspectos básicos da física de neutrinos. Descreve a história do neutrino desde sua concepção teórica até os dias atuais, apresentando, de forma sucinta, uma série de experimentos realizados.

O Capítulo 3 descreve a linha de feixe de neutrinos NuMI e o experimento MINERνA.

O Capítulo 4 descreve o *software* do experimento MINERνA .

O Capítulo 5 apresenta o trabalho que resultou no desenvolvimento de pacotes para o processamento e monitoramento em tempo real do experimento.

O Capítulo 6 descreve a instalação e testes com um protótipo do detector.

O Capítulo 7 faz um sumário e apresenta as conclusões desta tese.

Acrescentamos 5 apêndices: o apêndice A contém a estrutura de pacotes do *software* do MINERνA ; o apêndice B apresenta a estrutura do arquivo DST do MINERνA ; o apêndice apêndice C é a versao portuguesa do documento escrito para a colaboração com uma descrição do sistema de aquisição de dados; o apêndice D apresenta a versão portuguesa do manual de usuário do sistema quase-*online* do MINERνA ; o apêndice E apresenta todos os códigos desenvolvidos neste trabalho.

Capítulo 2

Física do Neutrino

2.1 História

Em 1911, um experimento realizado por von Bayer, Otto Hahn e Lise Meitner [3] sugeriu que a energia do elétron emitido no decaimento β apresentava um espectro contínuo ao invés de discreto. Isto estava em aparente contradição com a lei de consevação da energia já que, aparentemente, energia era perdida durante o processo de decaimento.

Em 1927 Ellis e Wooster [4] estabeleceram, sem sombra de dúvidas, que o espectro do decaimento β é, de fato, contínuo.

Em decaimentos radioativos, os núcleos dos átomos se transformam em núcleos diferentes quando o nêutron decai em um próton com emisão de eléctrons:

$$n \rightarrow p + e + \overline{\nu} \tag{2.1}$$

Sem o antineutrino, a conservação de energia requer que o eléctron e o próton tenham, em conjunto, a energia do nêutron. Cada eletron deve, por isso, ser produzido com energia fixa, enquanto os experimentos mostraram que os elétrons não são mono-energéticos, mas apresentam-se em uma faixa contínua de energia. Esta faixa de energia corresponde exatamente ao estado final de três partículas que podem compartilhar energia, satisfazendo as leis de conservação .

Numa carta para a conferência de físicos em Tubingen(Alemanha) em 1930[5] Wolfgang Pauli, dirigiu-se aos *Liebe Radioaktive Damen und Herren*[1]. Ele propõe um férmion que chamou de nêutron com massa da ordem de grandeza da massa do eléctron e com carga elétrica zero que também seria emitido no decaimento β.

[1]Prezados senhores e senhoras radioativos.

Quando Chadwick descobriu o nêutron, como o conhecemos hoje [6], em 1932, Fermi chamou de neutrino a partícula de Pauli, para diferenciá-la do nêutron pesado de Chadwick. A partir da comparação das formas observadas dos espectros, Fermi [7] e Perrin [8] postularam, em 1933, cada um, o neutrino sem massa.

Em 1934, Fermi[9] formalmente desenvolveu a teoria do decaimento β num sistema da eletrodinâmica quântica de Dirac, Heisenberg e Pauli.

Somente em 1956 Reines e Cowan[10] observaram diretamente o neutrino. Eles observaram a reação

$$\overline{\nu}_e + p \rightarrow n + e^+ \tag{2.2}$$

empregando um reator como fonte de anti-neutrinos de 1 MeV e um alvo de cloreto de cádmio e água. O detector tinha um grande volume e o reator fornecia um fluxo intenso de neutrinos, uma necessidade devido à baixíssima secção de choque deste processo.

Em 1958, Goldhaber determinou a helicidade do neutrino [11] verificando serem eles sempre levógiros. Em 1959 Davis demonstrou que ν podiam ser distinguidos de $\overline{\nu}$ [12]. Em 1960 um experimento conduzido por Lederman et. al. no acelerador AGS de Brookhaven [13] detectou um segundo tipo de neutrino: o ν_μ.

Em 1973 um experimento na câmara de bolhas Gargamelle do CERN descobriu interações de neutrinos na corrente neutra [14].

Experimentos com neutrinos solares iniciaram-se em 1968 por Davis [15] e observaram uma discrepância entre as previsões teóricas e o fluxo de neutrinos solares observado. Esta discrepância ficou conhecida como o problema do neutrino solar.

Experimentos realizados com neutrinos atmosféricos também apresentaram uma discrepância entre o fluxo de neutrinos observados e o fluxo esperado. Os experimentos observaram no fluxo de neutrinos atmosfericos, que os neutrinos muônicos aparentemente desaparecem em comprimentos de propagação da ordem de centenas de quilômetros. Experimentos que mediram o fluxo de neutrinos solares acharam resultados que sugeriam o desaparecimento de neutrinos eletrônicos, na propagação dentro do sol ou entre o sol e a Terra.

Este resultado levou Gribov e Pontecorvo[16], em 1968, a descreverem a oscilação de sabores caso os neutrinos tenham massa. O "desaparecimento"de neutrinos atmosfericos(ν_μ) e neutrinos solares(ν_e) não é fácil de explicar em termos de oscilação de neutrinos, a menos que pelo menos dois neutrinos tenham massa e se misturem. Isto não foi previsto pelo modelo padrão.

Em 1989, apenas alguns mêses após o início da tomada de dados com o novo acelerador LEP do CERN estudos da vida média do bóson Z revelaram a existência de três famílias de neutrinos [17, 18, 19, 20]. O terceiro neutrino, entretanto, só foi observado direitamente em 2000 pelo experimento DONUT [21].

Foi uma longa jornada desde os tempos pioneiros até o presente quando obteve-se prova de que os neutrinos tem massa. Um amplo caminho onde teóricos e experimentais caminharam de mãos dadas para mostrar a primeira evidência da física além do modelo padrão.

2.2 Propriedades dos neutrinos

Várias propriedades dos neutrinos já foram medidas, por exemplo:

- O experimento clássico de Wu et. al. [22] em 1957 determinou que as interações fracas violam a conservação de paridade. Este resultado, aplicado a neutrinos sem massa leva a conclusão de que neutrinos devem ser polarizados, com helicidade +1 ou -1. Entretanto, em 1958 Goldhaber et al [11] mediu a helicidade do neutrino e observou que apenas neutrinos levógiros participam da interação fraca;
- Há apenas três tipos de neutrinos com acoplamento padrão ao Z e com massas menores que 45 GeV/c^2 [17, 18, 19, 20];
- neutrinos satisfazem à conservação de número leptônico, o que significa que um neutrino está sempre associado a seu lépton carregado.
- As massas dos neutrinos não são conhecidas. Embora tenham sido tradicionalmente consideradas como zero, experimentos recentes indicam serem elas muito pequenas, porém não nulas. A observação de oscilações de neutrinos é um indicativo de sua massa ser não nula.

2.3 Experimentos realizados

Apresentamos nesta secção um resumo de vários experimentos realizados com neutrinos. Esta lista não se pretende completa, mas descreve experimentos importantes e cujos resultados foram fundamentais para a compreensão que hoje temos dos neutrinos.

2.3.1 Neutrinos atmosféricos

Neutrinos atmosféricos são produzidos, segundo uma abordagem simples, quando partículas de raios cósmicos interagem na atmosfera superior, produzindo um chuveiro secundário de mésons cujos decaimentos produzem neutrinos. Vários experimentos fornecem evidências para oscilações de neutrinos atmosféricos.

Super-Kamiokande

Consiste de um grande detector Čerenkov de água localizado na mina Mozumi no Japão. O detector está situado a uma profundidade de 1.000 m correspondendo a 2.700 metros equivalentes de água[2] (mea). Sua massa total é de 50 kt. Consiste de um tanque cilíndrico cheio com 680 t de água. O experimento detecta e mede neutrinos a partir do deslocamento do elétron na reação de espalhamento elástico:

$$\nu_x + e^- \rightarrow \nu_x + e^- \tag{2.3}$$

onde $x = e, \mu, \tau$. Embora sensível aos três sabores de neutrinos, a reação de espalhamento elástico é dominada pelo fluxo de ν_e O experimento Super-Kamiokande [23] encontrou evidências de oscilações de neutrinos ao registrar um déficit de $\nu_\mu(\overline{\nu}_\mu)$ enquanto observava o fluxo esperado de $\nu(\overline{\nu}_e)$.

Soudan2

Trata-se de um detector consistindo de um calorímetro com capacidade de rastreamento que foi instalado a 2090 mea em uma mina em Soudan, Minesota, EUA. Possuía massa total de 963 tons e foi construído para buscar decaimento de núcleons.

O experimento Soudan2 [24] observou o mesmo efeito de oscilação de neutrinos, de forma consistente com os resultados do experimento Super-Kamiokande. Note-se que os dois experimentos são diferentes, empregando materiais diferentes, tecnologias de detecção diferentes e técnicas diferentes de análise, o que reduz drasticamente a probabilidade de que erros sistemáticos possam falsear o sinal de oscilação de neutrinos.

SNO

O Observatório de Neutrinos de Sudbury (SNO) é um detector Čerenkov de 1 kt de água pesada (D_2O) ultra pura localizado a uma profundidade equivalente a 6.010 m de água na mina Sudbury, no Canadá. Sudbury mediu o fluxo de neutrinos solares2.3.1.

[2]Isto significa que a rocha envolvendo o detector equivale a colocar o detector sob 2.700 metros de água. É comum empregarmos esta unidade de medida de profundidade em física realizada em espaços subterrâneos.

MACRO

Tratava-se de um experimento localizado no Laboratório de Gran Sasso, Itália. O detector estava colocado a 3.700 mea. MACRO tinha como objetivo buscar monopólos e múons vindos de fontes cósmicas.

2.3.2 Neutrinos solares

As reações de fusão nuclear no sol produzem um fluxo intenso de neutrinos[3] . Essa reações podem ser representadas pelos processos:

$$\begin{aligned} p + p &\rightarrow {}^2H + e^+ + \nu_e \\ p + e^- + p &\rightarrow {}^2H + \nu_e \\ e^- + {}^7Be &\rightarrow {}^7Li + \nu_e \\ {}^8B &\rightarrow {}^8Be + e^+ + \nu_e \end{aligned} \tag{2.4}$$

O primeiro sinal persistente de oscilações de neutrinos foi observado em neutrinos solares. A detecção de neutrinos solares foi discutida, do ponto de vista experimental, por Davis [12]. Um número diferente de detectores persistentemente observava um número de neutrinos menor do que o esperado. Embora oscilação de neutrinos seja uma explicação possível para esta carência, não é a única. O número menor de neutrinos podia ser devido, por exemplo, a um modelo extremamente simples do Sol. Mohapatra [26] apresenta uma revisão histórica da situação experimental no final da década de 1990. O experimento SNO [25] observou uma taxa de neutrinos não eletrônicos (ν_μ,ν_τ) oriundos do Sol não nula. Como apenas ν_e são produzidos no Sol, isto constitui forte evidência para as oscilações de neutrinos. Evidências adicionais vieram do experimento KamLAND [27], no Japão, que emprega anti-neutrinos oriundos de um reator nuclear.

2.3.3 Experimentos com aceleradores

Experimentos deste tipo empregam um feixe de neutrinos produzidos em acelerador. Neste arranjo o feixe original, em geral de p, colide com um alvo. Mésons resultantes da colisão são selecionados por intermédio de "cones"magnéticos e enviados em direção ao detector de neutrinos, posicionado a uma certa distância. O decaimento destes mésons produz um feixe de neutrinos.

[3] A energia média nos neutrinos que emergem do sol é da ordem de $0,6$ eV

O experimento K2K [28] emprega um feixe de ν_μ com energia média de 1,3 GeV e observou 28 eventos quando esperava-se cerca de 38 eventos no caso de não ocorrerem oscilações.

O experimento MINOS [29] é um experimento de oscilações de neutrinos empregando o feixe de neutrinos NuMI do FERMILAB. Emprega dois detectores funcionalmente idênticos localizados a 1 km (o detector próximo) e a 750 km (o detector distante) da região onde os neutrinos são produzidos. O detector próximo mede a composição e o espectro do feixe de neutrinos enquanto o detector distante busca evidências de oscilações de neutrinos. MINOS observou um total de 215 eventos com neutrinos abaixo de 30 GeV comparado com 336,0 $\pm$ 18,3 $\pm$ 14.4 eventos que seriam esperados caso não ocorressem oscilações.

2.3.4 Experimentos com reatores

Vários experimentos foram realizados empregando reatores como fonte de anti-neutrinos. Um experimento empregando reator nuclear em 1956 propiciou a descoberta do neutrino [10]. Mais recentemente podemos citar o detector CHOOZ, na França. O detector tem um volume central de cintilador com massa de 5 t onde os $\overline{\nu}_e$ são detectados. Os resultados deste experimento [30] são consistentes com o desaparecimento de $\overline{\nu}_e$.

2.4 O Modelo Padrão

O Modelo Padrão é a base teórica da física de partículas. Consiste do que aparentemente seriam três teorias diferentes: eletromagnetismo, interações fracas e interações fortes[4]. Eletromagnetismo e interações fracas foram unificados nas interações eletrofracas.

O Modelo Padrão descreve a natureza em termos de partículas elementares divididas entre as que constituem os blocos elementares de matéria (quarks e léptons) e as partículas mediadoras das interações, chamadas de bósons intermediários.

Os quarks apresentam um número quântico chamado "cor" que aparece em três variedades denominadas de *vermelho*, *azul* e *verde*[5]

Os léptons não possuem "cor" e não interagem fortemente, enquanto cada um dos mediadores está associado a um tipo de interação: o fóton está associado com a interação eletromagnética; os bósons $W^{\pm}$ e Z^0 estão associados à interação fraca. O Modelo Padrão tem sido bem sucedido na explicação da física de partículas, entretanto neutrinos com massa não nula podem ser considerados como evidência de uma física além do Modelo Padrão[6].

[4] A gravitação não é faz parte do modelo padrão.

[5] O termo "cor" não apresenta qualquer relação com a definição usual de "cor", a qual está ligada à luz visível.

[6] No Modelo Padrão não existe o neutrino destrógiro, que permite o mecanismo para a massa

A tabela 2.1 apresenta algumas propriedades das partículas do Modelo Padrão.

QUARKS		
Quarks	Massa	Carga elétrica
up (u)	1,5 a 3,0 MeV/c^2	$+\frac{2}{3}$
down (d)	3,0 a 7,0 MeV/c^2	$-\frac{1}{3}$
strange (s)	95±25 MeV/c^2	$-\frac{1}{3}$
charm (c)	1,25±0,09 GeV/c^2	$+\frac{2}{3}$
bottom (b)	4,7±0,07 GeV/c^2	$-\frac{1}{3}$
top (t)	172,5±2,7 GeV/c^2	$+\frac{2}{3}$
LÉPTONS		
Léptons	Massa	Carga elétrica
elétron (e)	0,510998918 ± 0,000000044 MeV/c^2	-1
neutrino eletrônico (ν_e)	< 2 eV/c^2	0
muon (μ)	105,6583568 ± 0,0000052 MeV/c^2	-1
neutrino muônico (ν_μ)	$< 0,19$ eV/c^2	0
tau (τ)	1776,90 ± 0,20 MeV/c^2	-1
neutrino tauônico (ν_τ)	$< 18,2$ eV/c^2	0
BÓSONS INTERMEDIÁRIOS		
Bósons	Massa	Carga elétrica
fóton(γ)	$< 6 \times 10^{-17}$ eV/c^2	0
W$^\pm$	80,403 ± 0,029 GeV/c^2	±1
Z^0	91,1876 ± 0,0021 GeV/c^2	0
glúon (g)	0	0

Tabela 2.1: Partículas que compõem o modelo padrão de interações da natureza [31]

Neutrinos podem acoplar-se aos bósons portadores da interação fraca através de processos de corrente neutra (NC) ou corrente carregada (CC) como ilustrado na figura 2.1

Não há qualquer exigência intrínseca, no Modelo Padrão, que os neutrinos tenham massa zero. Entretanto, uma extensão do modelo é necessária para acomodar neutrinos com massa. Esta extensão pode ser conseguida pela inclusão de um neutrino destrógiro ν_R que não se acopla com os portadores da interação fraca, sendo , portanto, "estéril". A busca por tais neutrinos "estéreis" é uma importante ferramenta de teste do modelo padrão. Até o momento tais neutrinos não foram encontrados.

Há várias questões abertas quanto à física de neutrinos. Por exemplo:

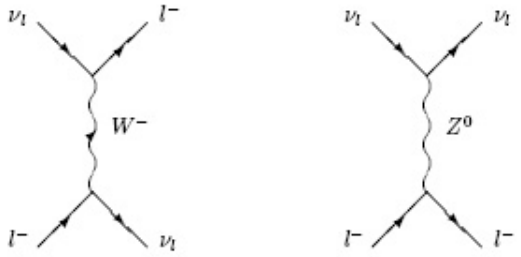

Figura 2.1: Diagramas de Feynman representando contribuições de corrente neutra (direita) e corrente carregada (esquerda) ao processo de espalhamento $\nu_l + l \rightarrow \nu_l + l$, onde l representa qualquer uma das três famílias de léptons.

- Quais são as massas dos neutrinos?
- Qual é o padrão de mistura entre os diferentes tipos de neutrinos?
- Os neutrinos são as suas própias antipartículas?
- Os neutrinos violam a simetria de CP?
- Existem neutrinos estéreis?
- O que os neutrinos podem nos dizer acerca dos modelos de nova física além do modelo padrão?

A resposta à maioria destas perguntas envolve o entendimento de como neutrinos interagem com a matéria. A baixíssima secção de choque dos neutrinos faz com que hajam poucos dados e que a física do neutrino ainda seja pouco conhecida. O advento de feixes intensos de neutrinos, com energia e fluxo conhecidos e controlados, como o NuMI do FERMILAB, abre a possilidade para experimentos de neutrinos com alta estatística o que permitirá estudos detalhados e precisos de forma até hoje impossível.

Capítulo 3

O experimento MINERνA

3.1 O feixe NuMI

A linha de feixe NuMI do FERMILAB foi projetada para produzir um feixe de neutrinos muônicos de alta intensidade. Um feixe de prótons de 120 GeV, extraído do anel de armazenagem do Main Injector, colide com um alvo de grafite produzindo várias partículas. Mésons carregados são focalizados por um sistema de magnetos focalizadores e, depois, decaem em uma área mantida em vácuo, produzindo neutrinos. A figura 3.1 mostra os principais componentes da linha de feixe NuMI. Uma descrição detalhada do feixe NuMI pode ser encontrada em [32] e [33].

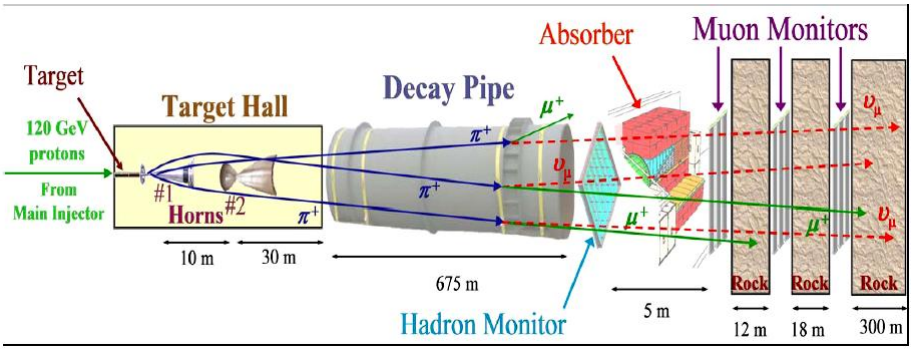

Figura 3.1: Principais componentes da linha de feixe NuMI. Mésons positivos, produzidos no alvo, são focalizados por dois cones magnéticos e decaem em uma região de 675 m mantida em vácuo, produzindo, predominantemente, ν_μ. Os hádrons restantes são bloqueados em um absovedor de feixe colocado ao final da região de decaimento.

A produção do feixe NuMi começa com a aceleração do feixe primário de prótons envolvendo três etapas: o LINAC, o "booster" e o "Main Injector". O LINAC acelera os prótons até 400 MeV e os repassa ao "booster" que acelera-os até 8 GeV. O "Main Injector" acelera os prótons

até atingirem a energia de 120 GeV.

Os prótons são, então, extraídos e enviados em direção ao alvo de grafite. Todo o processo dura cerca de $1,9$ s. As salvas de prótons resultantes contêm cerca de $2,5 \times 10^{13}$ prótons e tem duração de cerca de $9,7$ μs.

O alvo de grafite mede $0,95$m de comprimento, é segmentado e refrigerado a água. O alvo tem espessura de apenas $6,4$ mm, para minimizar reabsorção, e representa $1,9$ comprimentos de interação hadrônica para maximizar a produção de mésons.

Os mésons positivos criados no alvo são focalizados por um par de dispositivos magnéticos de focalização chamados de "cornetas" ("horns"). As cornetas do NuMI são como lentes magnéticas parabólicas que criam um campo toroidal com valor de pico de 3 T. Elas são operadas em modo pulsado com uma corrente nominal de 200 kA e resfriadas por pulverização de água [32].

Após a focalização, os píons e káons entram em um cilindro, em vácuo, onde decaem produzindo um μ e um ν_μ. Esta região de decaimento consiste em um cilindro de 675 m de comprimento, 2 m de diâmetro e é mantido à pressão residual de, aproximadamente, 1 torr ou menor.

Hádrons que cheguem ao final desta região são parados por um absorvedor de feixe, que consiste de um núcleo de alumínio refrigerado à água envolvido por uma camada de blocos de aço e uma camada externa de concreto[1].

O feixe de neutrinos resultante consiste de $97,8\%$ ν_μ e possui poucos $\overline{\nu}_\mu$ $(1,8\%)$ e ν_e $(0,4\%)$

Uma das características do feixe NuMI está na capacidade de ajuste do espectro de energia do feixe de neutrinos. A figura 3.2 mostra três diferentes configurações possíveis: baixa energia (LE), média energia (ME) e alta energia (HE).

As energias diferentes são obtidas movendo-se o alvo e/ou a segunda corneta focalizadora, de forma similar ao movimento de lentes em um sistema óptico. Para que isto possa ocorrer, o alvo está montado em um sistema de trilhos que permite movê-lo ao longo do feixe por uma distância de até $2,5$ m. Com isto, píons e káons de momenta diferentes são selecionados e focalizados na região de decaimento, resultando em diferentes espectros de energia dos neutrinos.

A polarização das "cornetas" pode mudar de tal manera que os mesons focalizados sao do signal oposto, permitendo ao Numi produzir feixes de neutrinos e anti-neutrinos.

A linha NuMi fornece um feixe intenso de neutrinos para o experimento de oscilação de neutrinos MINOS. O *Near Detector* do MINOS[2] está instalado em uma grande área subterrânea (a 100 m de profundidade) construída no FERMILAB. Esta área possui espaço suficiente para que seja instalado um experimento de espalhamento ν e $\overline{\nu}$-nucleon de alta estatística e alta

[1] Os múons, porventura ainda existentes, são parados nos 300 m de rocha que separam a área experimental do absorvedor.

[2] nome usado por ser o detector mais proximo ao inicio do feixe NuMi

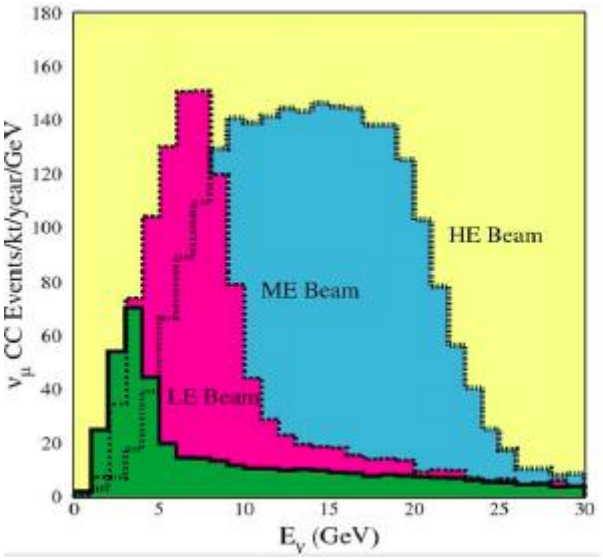

Figura 3.2: Diferentes configurações do feixe NuMI. O espectro do feixe NuMI pode ser alterado movendo-se o alvo e/ou a segunda corneta focalizadora.

resolução. O experimento MINERνA, que é objeto desta tese, terá seu detector instalado imediatamente à montante do *Near Detector* do MINOS e medirá secções de choque de neutrinos e verificará os efeitos nucleares com a dependência do número atômico. A alta intensidade do feixe NuMI permite que possamos investigar pela primeira vez diversos aspectos de uma grande variedade de assuntos importantes na física de neutrinos (ou aumentar significativamente o conhecimento em áreas até aqui pouco investigadas ou com pouca estatística).

As figuras 3.3 e 3.4 mostram a área onde será instalado o detector MINERνA .

3.2 Descrição geral

Para realizar todos os estudos a que se propõe, o detector do MINERνA precisa estabelecer um novo paradígma no projeto de experimento de neutrinos[3]. Com estados finais tão variados como reações inelásticas com alta multiplicidade, produção coerente de píons e espalhamento quase-elástico[4] de neutrino, o detector é um híbrido de um calorímetro tradicional com um detector ativo[5] altamente segmentado. O detector deve ser capaz de:

- Identificar múons e medir seus momenta com grande precisão.

[3]Uma apresentação mais detalhada do detector pode ser encontrada em [34].

[4]Em física de neutrinos o termo quase-elástico é empregado para descrever um processo no qual um neutrino interage com um nucleon e produz um lépton carregado no estado final. O termo refere-se ao fato de que o neutrino do estado inicial transforma-se em um lépton diferente e há apenas um único nucleon no estado final.

[5]Chamamos de detector ativo quando a mesma estrutura exerce as funções de alvo e detecção simultaneamente.

Figura 3.3: Vista parcial da caverna onde está colocado o detector do MINERνA . Este espaço encontra-se a 100 m de profundidade.

- Identificar hádrons e π^0 individuais e medir seus momenta.
- Medir a energia de chuveiros hadrônicos e electromagnéticos com uma melhor precisão.
- Minimizar a "confusão" entre eventos de corrente-neutra e de corrente-carregada.
- Acomodar diversos alvos nucleares.

Estes objetivos podem ser alcançados com um detector/alvo compacto consistindo, basicamente, de barras cintiladoras. O núcleo do detector MINERνA consiste de um detector constituído de barras de cintilador sólido. Este detector plástico interior, conhecido como *Inner Detector* (ID) serve como o volume primário onde um rastreamento preciso de partículas é possível. Um material de baixa densidade e amostragem fina asseguram que algumas das medidas mais difíceis possam ser feitas. Isto inclui, entre outras, rastreamento de fótons, deteção de prótons de recuo em eventos quase-elástico de baixo Q^2 e identificação de partículas através de dE/dx. A figura 3.5 mostra uma vista lateral do detector.

Figura 3.4: Vista da parcial da área onde ira localizar-se o detector do MINERνA . A estrutura visível ao fundo é o detector próximo do MINOS. O detector do MINERνA ocupará o espaço antes deste detector, que servirá de detector auxiliar do MINERνA na identificação de múons.

O detector central é cercado, por todos os lados, pelos calorímetros electromagnético, hadrônico e um identificador de múons. O detector tem forma aproximadamente hexagonal com secção reta variando de 3,35 m a 4,10 m. O comprimento é de 5,9 m. A massa total de cintiladores é de cerca de 6 toneladas. À montante do detector encontram-se os alvos nucleares consistindo de, aproximadamente, 0,7 kg de Fe, 850 kg de Pb e de 200 kg de He líquido. O detector apresenta significativa granularidade, o que assegura precisão na reconstrução de vértices graças ao uso de cintiladores plásticos em forma de prismas triangulares com 3,3 cm de base, 1,7 cm de altura e comprimento de cerca de 4,0 m, com uma fibra ótica inserida em sua base para leitura do sinal. Barras semelhantes são empregadas nos calorímetros.

Nas baixas energias necessárias para estudo de seções de choque, muitos eventos contêm partículas que diringem-se para as laterais e para trás. Assim, os detectores de amostragem extendem-se para os lados, e mesmo para trás, do detector. Múons energéticos penetram no detector próximo do MINOS onde seus momenta podem ser medidos. Assim, o detector próximo do MINOS opera como um detector auxiliar do MINERνA.

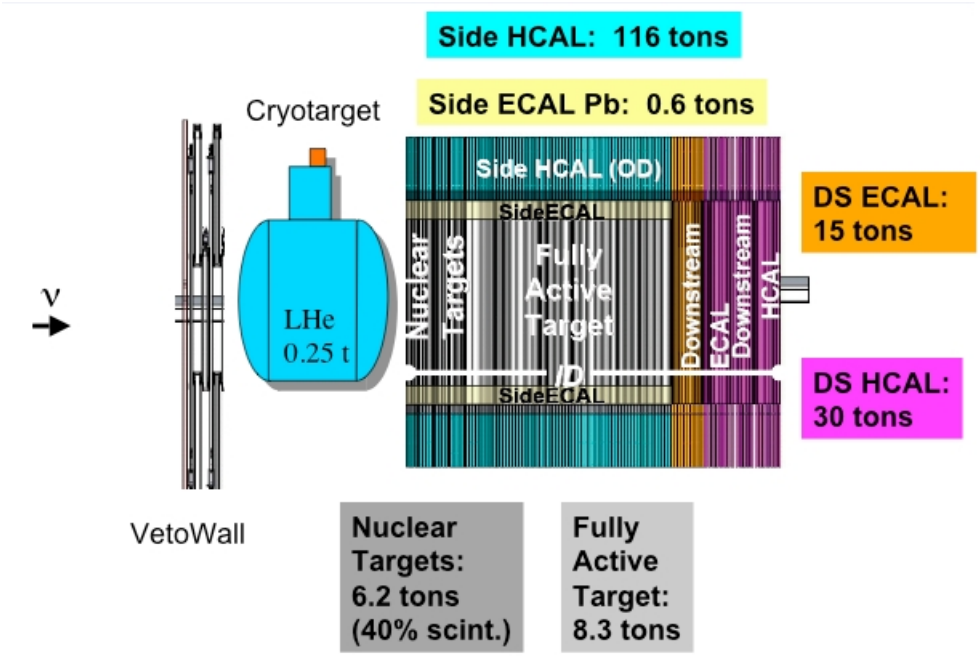

Figura 3.5: Vista esquemática lateral do detector do MINERνA

Exceto pelo veto à montante, o Detector do MINERνA é segmentado tranversalmente no ID com planos de barras cintiladoras sólidas. Por construção e conveniência, um plano do MINERνA incorpora as estruturas do ID e do OD, além da estrutura externa de suporte. Dois planos de cintilador são montados em uma estrutura, chamada de modulo como ilustra na figura 3.6. Há 3 distintas orientações no ID, deslocadas de 60^o e chamadas X, U e V. As diferentes orientações permitem uma reconstrução tridimensional de trajetórias e o deslocamento de 60 graus faz do hexágono a forma natural da secção reta transversal do detector.

A figura 3.7 mostra uma vista de frente do detector olhando-se na direção de onde vem o feixe. A figura humana ilustra as dimensões do detector.

O centro do detector é o *Inner Detector*(ID) (totalmente ativo), cujo núcleo plástico representa o volume efetivo para a maioria das análises do MINERνA . A figura 3.8 ilustra os módulos da região ativa central. Detectores calorimétricos na região central do detector são construídos com a inserção de absorvedores entre planos adjacentes. A figura 3.9 ilustra um módulo do calorímetro eletromagnético.

Placas de chumbo de 0,2cm de espessura são inseridas entre os planos de cintilador e na frente de cada modulo, para atuar como calorímetro eletromagnético lateral. Isto representa a maior parte do comprimento do detector, e o *Outer Detector*(OD) representa a maior parte da massa do detector.

O Inner Detector é envolvido por sistemas de absorvedores e cintiladores que fazem o *Outer Detector*(OD) . O Outer Detector compõe-se de seis “torres” (um sexto do hexágono).

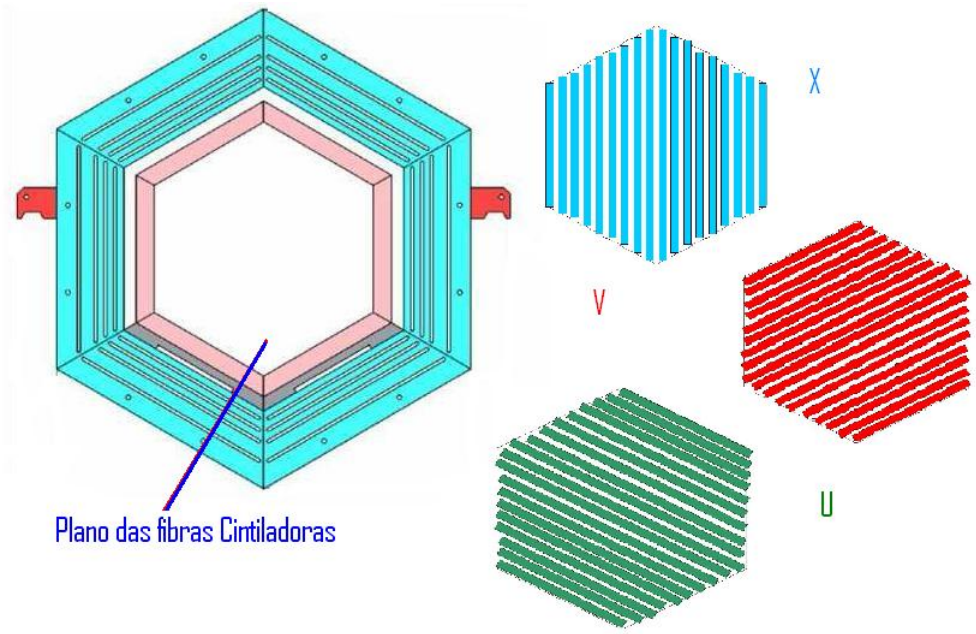

Figura 3.6: Vista de um módulo ativo do detector. A figura à direita mostra os três planos de fibras cintiladoras U,V e X deslocados de ±60 graus.

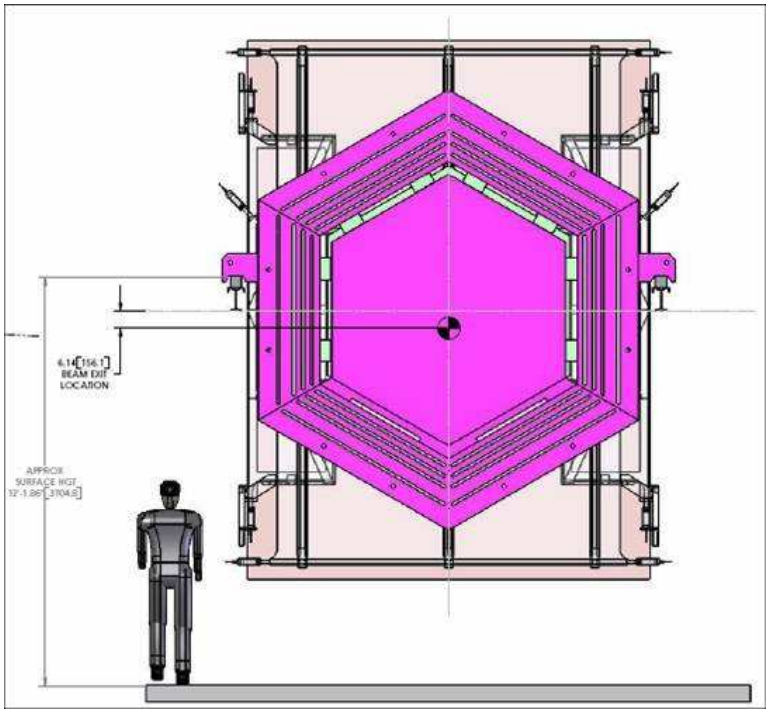

Figura 3.7: Vista do detector do MINERνA ilustrando forma e tamanho. A estrutura alta, parcialmente encoberta pelo detector, é a parede de veto

A parte mais à jusante do detector tem os calorímetros hadrônicos (HCALs) com absorvedores de 1 polegada por plano cintilador como mostrado na figura 3.10. Os calorímetros

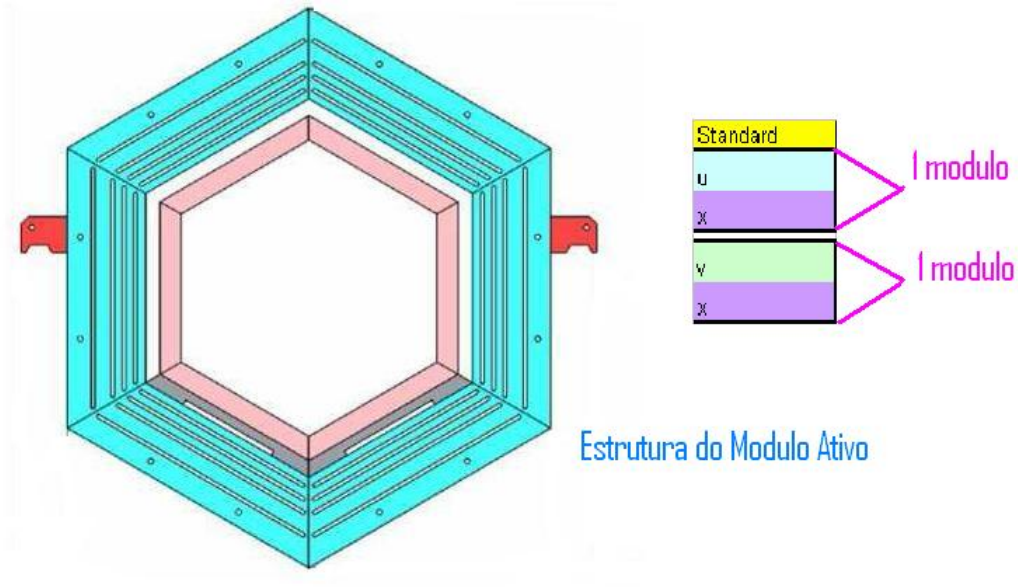

Figura 3.8: Vista de um módulo ativo do detector. A figura à direita mosta a estrutura deste tipo de módulo, UX ou VX

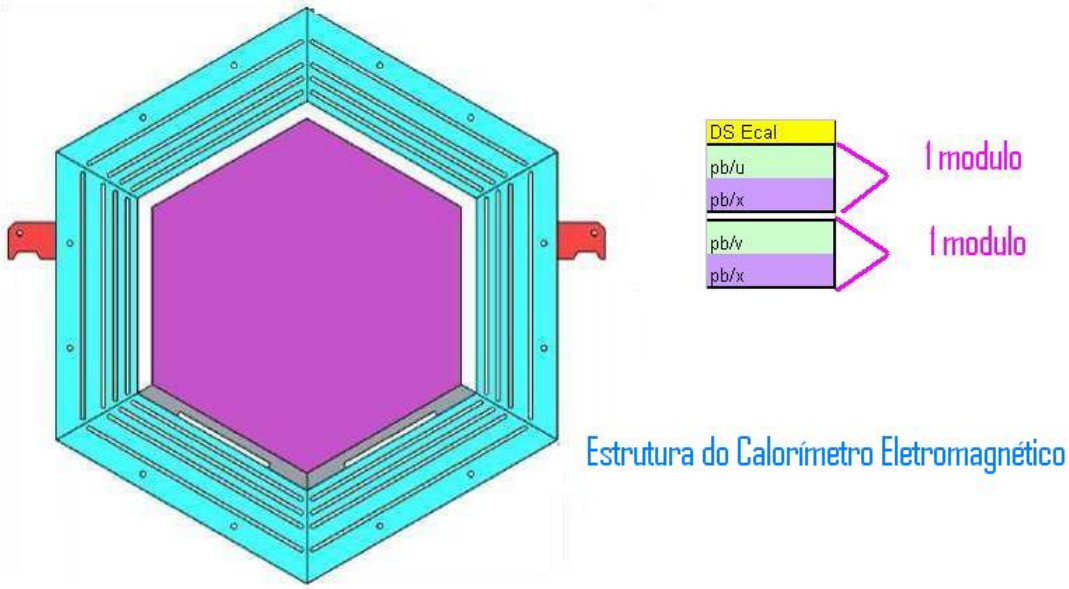

Figura 3.9: Vista de um módulo do calorímetro eletromagnético. A figura à direita mostra a estrutura do calorímetro, alternando capas de chumbo com capas cintiladoras.

eletromagnéticos possuem absorvedores de Pb de 0,2 cm por plano. A alta granularidade do ECAL assegura uma excelente resolução de energia de fótons e elétrons além de permitir a

medida da direção de ambos.

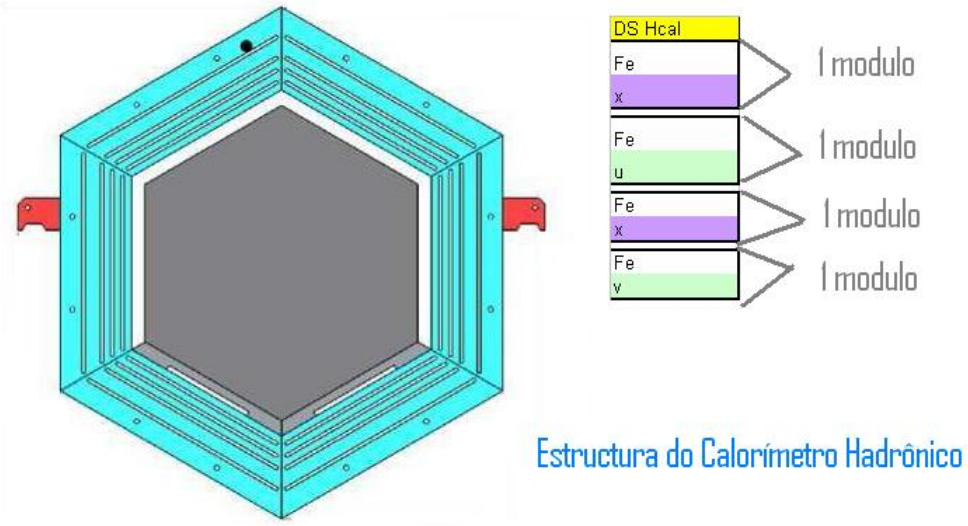

Figura 3.10: Vista de um módulo do calorímetro hadrônico. A figura à direita mostra a estrutura do calorímetro, alternando capas de ferro com capas cintiladoras.

Por fim, a região assinalada como Alvos Nucleares (*nuclear targets*) na figura 3.5 comtêm absorvedores colocados entre alvos ativos de forma que é possível o estudo de eventos em diferentes alvos nucleares. Estes alvos são descritos na secção 3.5.

3.3 Os dispositivos fotossensores

O custo maior do detector do MINERνA vem dos dispositivos fotossensores e eletrônica associada. No MINERνA o sinal de luz é suficiente para empregar fotocatodos com eficiência quântica de 15% e fibras de diâmetro de 1,2 mm. MINERνA emprega fotomultiplicadoras multi-anodo HAMAMATSU H8804 com 64 canais.

3.4 Os elementos cintiladores

No *Inner Detector*(ID) , os elementos sensíveis do MINERνA são cintiladores em forma de prismas triangulares, com 1,7 cm de altura por 3,3 cm de base feitos de polistireno (Dow 663) e dopados com POP (1% por peso) e POPOP (0,03% por peso) revestidos com uma cobertura reflectiva de TiO_2 com fibras ópticas alteradoras de comprimento de onda (WLS) (Kurrary Y11 com dopante 175 ppm) de 1,2 mm incrustadas(figura 3.11). As fibras WLS são levadas a conectores ópticos nas extremidades dos módulos e fibras ópticas claras levam a luz

até fotomultiplicadoras multianodo. Para melhorar a resolução de coordenadas, estes elementos triangulares são montados em planos (figura 3.12 e figura 3.13); isto permite repartir carga entre tiras de cintiladores vizinhas num simples plano para obter-se, por interpolação, a coordenada.

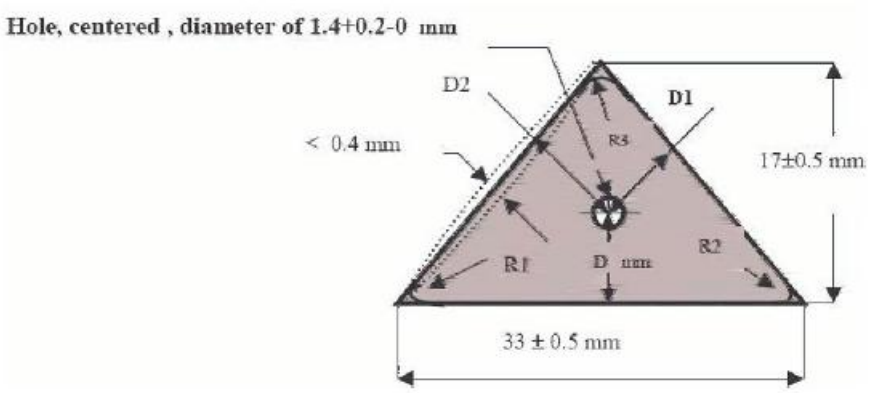

Figura 3.11: Corte transversal de um prisma triangular cintilador empregado no *Inner Detector*. Nota-se o furo por onde a fibra óptica é inserida.

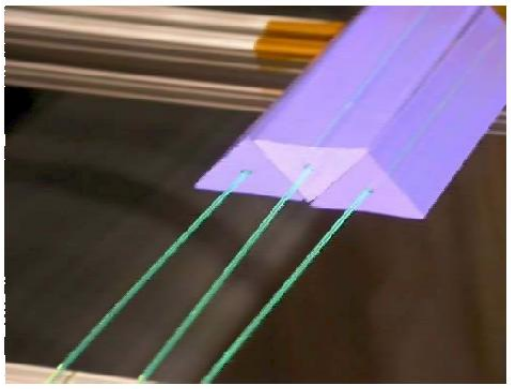

Figura 3.12: Prismas triangulares cintiladores arranjados de forma a constituirem um plano cintilador. Cada prisma corresponde a um canal. Cada prisma tem uma fibra óptica inserida ao longo de seu comprimento.

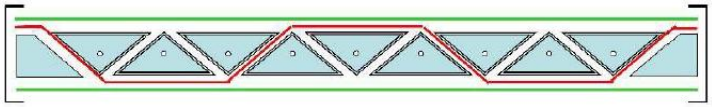

Figura 3.13: Vista de corte de um plano montado com prismas triangulares cintiladores.

3.5 Calorímetros e alvos

Os objetivos do MINERνA requerem a realização de medidas de energias de partículas carregadas (p,$\pi^{\pm}$,$K^{\pm}$,$\mu^{\pm}$) e neutras (π^0,K^0) com energias de até poucos GeV. Para atingir estes objetivos o MINERνA possui os seguintes elementos: um conjunto de chumbo e cintilador alternados à jusante e envolvendo o detector ativo para calorimetria electromagnética, um conjunto de aço e cintilador alternados à jusante e na parte exterior do detector para calorimetria hadrônica, e placas de chumbo, aço e carbono à montante do detector central para calorimetria electromagnética e hadrônica. Os calorímetros estão concebidos de forma a que chuveiros originados na região central sejam completamente contidos no sistema.

3.6 Calorímetro eletromagnético

A detecção de fótons de alta energia dá-se através do processo de produção de pares que leva a um chuveiro de e^+, e^- e γ. Como a seção de choque de produção de pares é proporcional a Z^2, folhas de chumbo são, em geral, utilizadas para produção de um chuveiro de tamanho razoável. O comprimento característico dos chuveiros varia com a energia. Porém, para fótons de até uns poucos GeV, como esperado em nosso experimento, 99% da energia estará contida em 4 cm de Pb (correspondente a 7 comprimentos de radiação).

O calorímetro electromagnético à jusante consiste de 20 camadas de Pb de 2 mm de espessura, alternadas com camadas de cintiladores feitas a partir dos cintiladores em forma de prisma triangular já mencionados. A resolução de energia esperada é de $6\%/\sqrt{E}$, com E dado em GeV. A figura 3.9 ilustra um módulo do ECAL. O calorímetro lateral é similar. Folhas de chumbo, também de 2 mm de espessura, são alternadas com camadas de cintiladores. Fótons que penetrem o calorímetro em ângulos de até 25^0 em relação a linha do feixe, são absorvidos. Para ângulos maiores eles não serão totalmente contidos, mas penetrarão no calorímetro hadrônico externo, onde o chuveiro restante está totalmente contido.

Como o objetivo principal das placas de Pb, Fe e C à montante é servir como alvo, a calorimetria nesta região não é tão eficiente quanto nos módulos à jusante e nas laterais. A amostragem é mais grosseira pois as placas de Pb, Fe e C são mais espessas. O arranjo dos alvos implica que o número de comprimentos de interação visto pelo chuveiro nesta região, antes de escapar do detector, varia entre 5 e 10. Entretanto, uma vez que os fótons nesta direção terão energia muito menor, os chuveiros que se iniciem na região central também estarão totalmente contidos.

3.7 Calorímetro hadrônico

O calorímetro hadrônico à jusante está posicionado logo após o calorímetro eletromagnético e consiste de 20 camadas de ferro de 2,54 mm de espessura alternadas com placas cintiladoras. A ação combinada de 4 cm de Pb e 50 cm de Fe pára múons de até 600 MeV e prótons de até cerca de 800 MeV. O comprimento de interação nuclear do Fe é 16 cm. Assim, prótons (ou píons) de maior energia também serão, provavelmente, detidos. A figura 3.10 ilustra um módulo do HCAL. O calorímetro hadrônico lateral consiste de uma placa de ferro de 55,9 cm de espessura, com 5 aberturas, cada uma com 2,5 cm, preenchidas com cintiladores. A espessura total de ferro é 43,4 cm, que é suficiente para parar prótons de até 750 MeV que penetrem a 90^o e prótons de até cerca de 1 GeV que penetrem a 30^o.

A resolução do calorímetro hadrônico é esperada ser cerca $50\%\sqrt{E}$ para hádrons com energia acima de 1 GeV. A resolução para partículas de menor energia e esperada ser 50% ou menos, dependendo da energia.

3.8 Alvos nucleares

Os alvos nucleares do MINERνA consistem de carbono, ferro e chumbo além de He. O Ferro foi escolhido por ser um material barato e por ser um absorvedor comum em experimentos com neutrinos, como o MINOS. Chumbo é o material nuclear mais alto que é de fácil obtenção. Idealmente os alvos nucleares deveriam ser feitos de muitos alvos finos com várias camadas rastreadoras entre cada alvo de forma a determinar-se a multiplicidade de estados finais e a quantidade de energia de cada uma das partículas de baixa energia. Alguns fatores, entretanto, limitam o número e tamanho dos alvos assim como o número de camadas rastreadoras. A resolução espacial intrínseca do detector é da ordem de 1 cm na direção do feixe. Assim, alvos mais finos seriam ineficientes. Placas com espessura maior que 2,5 cm (como as usadas no MINOS) não permitiriam uma melhora significativa do conhecimento do espectro de partículas de baixa energia, que é um dos objetivos do MINERνA. Para aumentar a estatística deveríamos ter cerca de 1 tonelada de cada alvo. Restrições práticas, entretanto, limitam o tamanho do alvo que podemos implementar. O MINERνA decidiu implementar um alvo seguindo o seguinte esquema onde cada F indica uma estrutura XU ou XV (um par FF é um conjunto XUXV).

FF Pb/Fe [alvo 1] FFFF Pb/Fe [alvo 2] FFFF Pb/Fe/C [alvo 3] FFFF Pb[alvo 4] FFFF Pb/Fe [alvo 5]

Alvos 1 e 2 tem 60% de Fe e 40% de Pb representando cerca 230 kg de Fe e de Pb em cada alvo. O alvo 3 tem 50% de C, 30% de Fe e 20% de Pb representando cerca de 140 kg de C e 110 Kg de Fe e 110 kg de Pb. Alvo 4 é de Pb puro com massa total de 170 kg. Alvo 5 tem

Pb e Fe, sendo 60% de Fe e 40% de Pb cada um com 115 kg. No total temos 685 kg de Fe e 855 kg de Pb. O número total de eventos que vão ocurrir nos alvos nucleares de CC (corrente carregada) é de: 2 milhões para o Fe, 2,5 milhões para o Pb e 400.000 para o C. Ilustrações da configuração dos alvos podem ser vistas na figura 3.14.

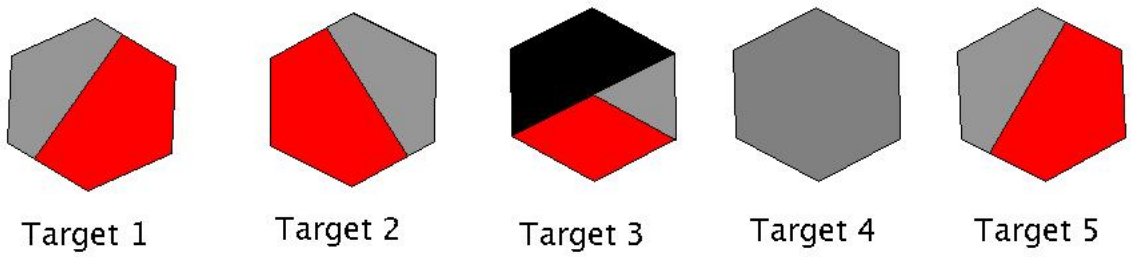

Figura 3.14: Alvos nucleares: Vermelho = Fe, Cinza = Pb, Preto = C

3.9 Eletrônica e aquisição de dados (DAQ)

A tabela 3.1 sumariza as exigências para a eletrônica do MINERνA . Estas exigências são motivadas pelos seguintes objetivos:

- Resolução espacial fina explorando o compartilhamento de luz entre barras cintiladoras vizinhas;
- Identificação de $\pi^{\pm}$ e p através da informação de dE/dx;
- Reconhecimento de padrões eficientes, empregando "timming" para identificar a direção da trajetória e separar interações que ocorram dentro do mesmo ciclo (*spill*);
- Habilidade para identificar partículas estranhas e decaimento de múons empregando técnicas de coicidência;
- Tempo morto desprezável dentro de um ciclo (*spill*).

A taxa média de dados esperada para o MINERνA é de aproximadamente 100 kBytes/s.

As exigências para o DAQ do MINERνA são modestas devido à taxa relativamente baixa de eventos. O tempo previsível do feixe torna um sistema de gatilho complexo desnecessário. No lugar de um tal sistema, simplesmente temos um sinal de *gate* que se abre imediatamente antes da chegada do feixe e todas as informações de carga e tempo de todo o detector são simplesmente registradas após o término do *spill*.

Parâmetro	valor
ciclo (*spill*)	12 μs
Tempo de repetição	>1,9 s
Número de canais	30.972
Ocupação por ciclo	2%
Variação de ganho do detector fotossensível	4,5 dB
Resolução temporal	3 ns

Tabela 3.1: Algumas das exigências e parâmetros para o projeto da eletrônica do MINERνA

3.9.1 Hardware do DAQ

Os elementos ativos do detector do MINERνA tem seus sinais enviados para fotomultiplicadoras multianodo (MAPMT) de 64 canais. A informação de amplitude e tempo dos sinais das MAPMT é digitalizada pela eletrônica e guardada para leitura pelo sistema de aquisição de dados (DAQ). Cada placa eletrônica de leitura (FEB) é conectada a uma única fotomultiplicadora.

Grupos de até 10 FEB são lidos e o resultado enviado para um módulo controlador de leitura de bastido (CROC - *Crate Read-Out Controler*) montado em um bastidor VME. Cada CROC pode acomodar 4 cadeias de litura de FEB. Um total de 12 CROCs são necessários para todo o detector do MINERνA . Os bastidores VME tambem alojam um módulo de interface do CROC (CRIM (*CROC Interface Module*), um módulo de tempo (MTM - MINERνA *Timing Module*) e uma fonte de 48 V. Não há CPU nos bastidores VME.

Durante a tomada de dados, o sistema DAQ funciona durante todo a salva de neutrinos. Ao final do períiodo de 10 μs o DAQ lê todos os canais que se apresentem sinal acima de um certo pedestal. Mesmo com alta taxa de ocupação o número total de *bytes* lidos a cada salva de neutrinos é menor que 200 kB com supressão de zeros (1 Mb sem supressão de zeros). O tempo morto é desprezável.

A fonte de alimentação de 48 V alimenta as bases das fotomultiplicadoras. MINERνA emprega o mesmo *hardware* para aquisição de dados e para para o sistema de controle do detector (DCS - *Detector Coontrol System*). A mesma conexão que faz a leitura dos FEB também fornece o canal de comunicação e controle do detector como, por exemplo, o controle das voltagens dos MAPMT. Os computadores principais para o DAQ e para o sistema de controle lento (o sistema de controle e monitoramento de variáveis cujo valor sofra alterações lentas) localizam-se próximos da eletrônica VME conectados à rede do FERMILAB através de duas linhas de alta velocidade TCP/IP. Uma servidora de duas CPU realiza estas tarefas tendo uma CPU dedicada à aquisição de dados e a outyra ao controle e monitoramento. As

máquinas do sistema DAQ rodam Windows e uma máquina rodando Linux encontra-se entre os computadores do DAQ e a rede geral.

3.9.2 *software* de Aquisição de dados (DAQ)

O software do MINERνA é montado no arcabouço GAUDI empregando a infraestrutura cliente-servidor GAUCHO, originalmente desenvolvidos para a colaboração LHCb. A taxa média de dados esperada para eventos sem supressão de dados é de apenas 100 kB/s e uma janela de dois segundos é disponível após cada salva de neutrinos de 10 μs. A alta previsibilidade do feixe torna desnecessário o estabelecimento de um sistema complicado de gatilho. No lugar de tal sistema, o MINERνA simplesmente ativa a eletrônica do detector e toda informação é lida ao final de cada salva. O sistema de controle lento é, igualmente, simples tendo cada MAPMT sua própria fonte de alta tensão local e os módulos FEB fazem a leitura das altas tensões, temperaturas e outros parâmetros empregados para monitoração e controle. Um diagrama esquemático é visto na figura 3.15.

3.10 Parâmetros do detector do MINERνA

A tabela 3.2 apresenta o número total de canais do detector do MINERνA em seus subdetectores. Os canais são dominados pela granularidade nos alvos de plástico, Pb e Fe. Quanto a massa, o domínio cabe ao *Outer Detector(OD)*. O MINERνA emprega 19,2 toneladas de cintiladores, 93 km de fibras ópticas cintilantes e outros 43 km de fibras ópticas claras entre o detector e 587 fotomultiplicadoras multianodos de 64 canais cada.

Sub-detector	ID	OD	massa (toneladas)
Alvo ativo e ECAL	15.360	5.760	14,6
ECAL à montante (Alvo de Pb)	3.072	1.152	3,5
HCAL à montante (Alvo de Fe)	1.536	576	7,0
ECAL à jusante	2.560	960	19,8
HCAL à jusante	2.560	960	26,4
Veto	426	n.a.	15,1
OD envolvendo as regiões do alvo	n.a.	n.a.	126,5
Total	25.514	8.408	211,3

Tabela 3.2: Parâmetros do detector do MINERνA.

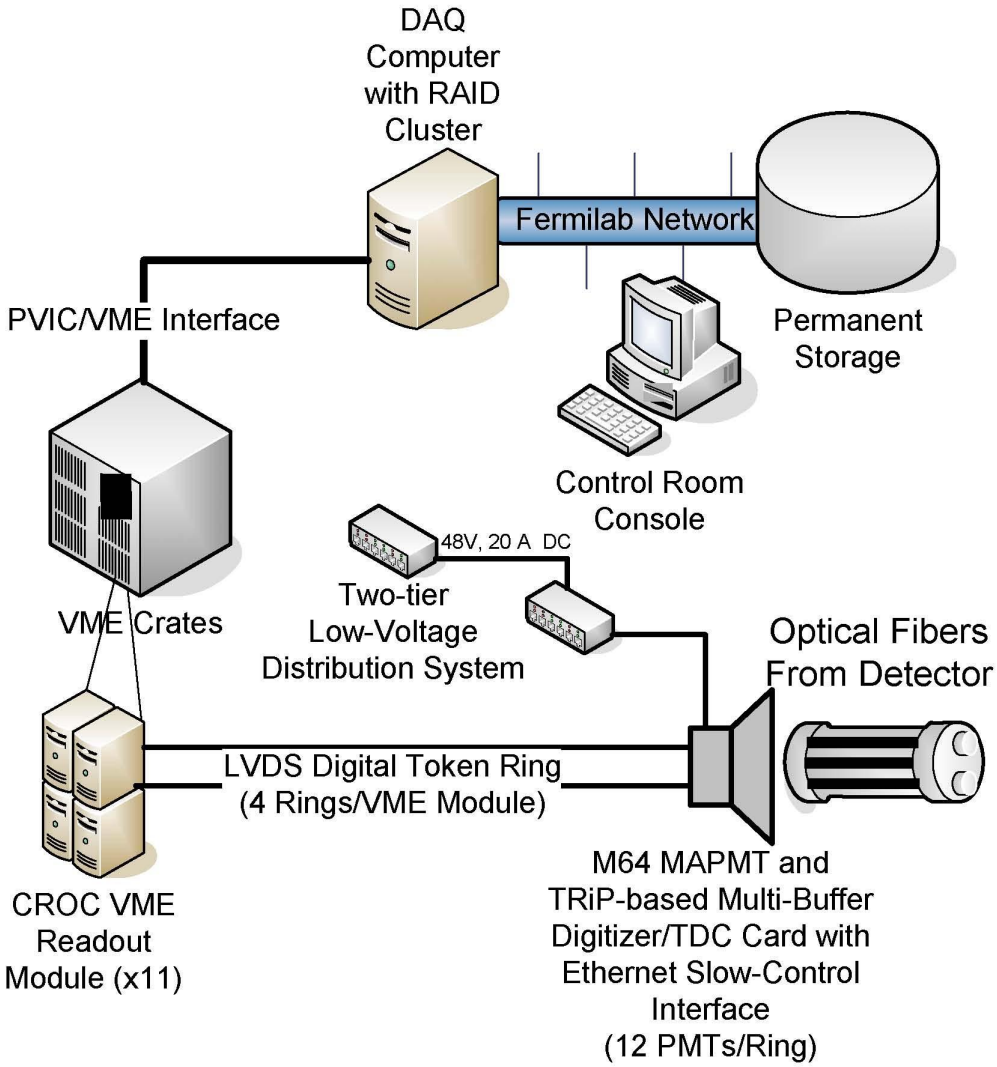

Figura 3.15: Diagrama esquemático do sistema de aquisição de dados do MINERνA

3.11 Testes

Vários subprojetos ocorrem paralelo à construção do detector do MINERνA . Realçamos aqui quatro deles: o mapeador de módulos, o protótipo do detector de rastreamento, uma estrutura de teste das MAPMT e um detector de prova.

3.11.1 Mapeador

O mapeador é um sistema que varre um módulo cintilador do detector com duas fontes radioativas e lê a resposta da fotomultiplicadora. Os resultados obtidos em cada pedaço do módulo durante cada passagem das fontes radioativas cada são digitalizados e armazenados. O DAQ incumbe-se de acumular os dados e de controlar o movimento das fontes. Este processo permite-nos encontrar variações ao longo das tiras cintilantes e realizar um mapeamento completo de cada um dos módulos.

A figura 3.16 ilustra o mapeador isolado e sobreposto a um módulo.

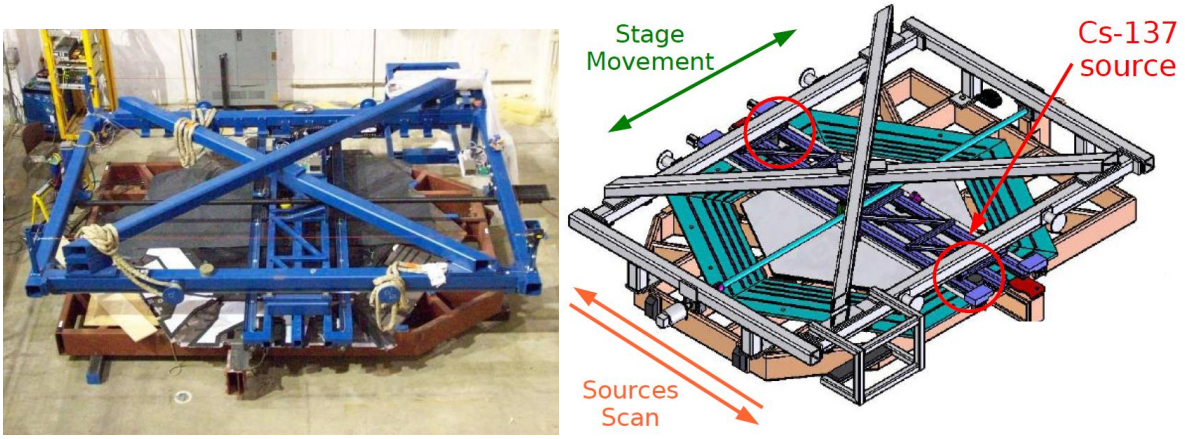

Figura 3.16: Mapeador.

3.11.2 Teste das fotomultiplicadoras - Forest Test

Grupos de MAPT passam por testes intensivos onde pulsos de luz são enviados a cada um dos elementos da fotomultiplicadora em diversas configurações. O sinal das fotomultiplicadoras são digitalizados e armazenados. Estes dados são usados para levantamento completo das características de todas as MAPMT usadas no detector. Este sistema permite determinamos parâmetros como ganho, linearidade e interferência entre canais. A estrutura montada permite o teste simultâneo de 19 fotomultiplicadoras.

A figura 3.17 ilustra a estrutura empregada para os rtestes das MAPMT.

3.11.3 Detector de prova

Este detector é uma versão do detector do MINERνA em menor escala e sem os alvos nucleares e será exposto a um feixe hadrônico.

A função deste detector é fornecer uma descrição clara da resposta do detector do MINERνA a partículas específicas. É fundamental para a calibração do detector do MINERνA . Empregará

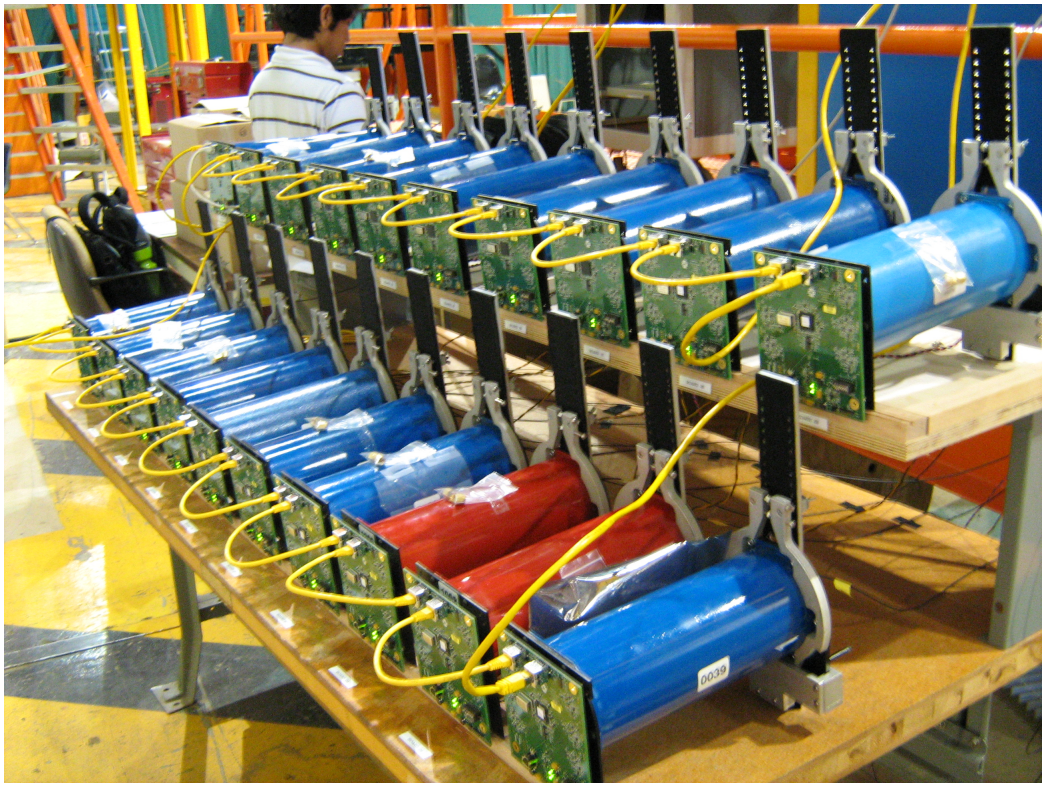

Figura 3.17: Montagem de teste das MAPMT (Forest Test).

feixe de hádrons com energia entre 200 MeV e 1 GeV.

Este detector é uma réplica em pequena escala do MINERνA e será construído com peças mais curtas dos mesmos cintiladores empregados no MINERνA , usará as mesmas MAPMT e eletrônica . O objetivo principal é verificar como o MINERνA responde a partículas individuais, especialmente prótons e píons. Isto inclui informação sobre a energia total de chuveiros hadrônicos, dE/dx ao longo das trajetórias, resolução de energia para estas quantidades, topologia da formação dos chuveiros e identificação de partículas. As informações obtidas por este detector de prova servirão de entrada para ajuste da simulação do detector (que emprega GEANT4) e para calibração da energia durante a fase de reconstrução. É, também, importante para o desenvolvimento de padrões de reconhecimento, medida das eficiências. Os padrões de espalhamento e absorção de píons em núcleos não são bem modselados nas simulações de Monte Carlo atuais.

O detector de provas do MINERνA contém 40 planos com orientação XUXV como no detector completo do MINERνA . A figura 3.18 ilustra o detector de prova. As tomadas de dados no detector de prova devem ocorrer no segundo semestre de 2009.

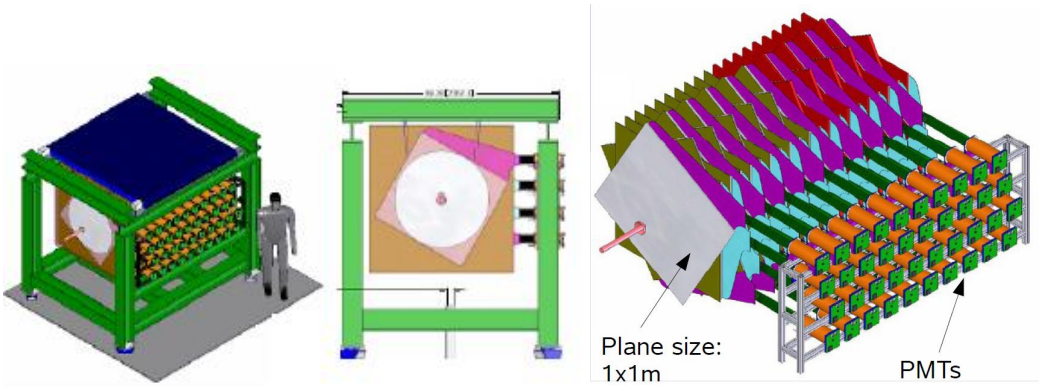

Figura 3.18: O detector de provas do MINERνA .

3.11.4 Protótipo de rastreamento

O protótipo de rastreamento consiste em 20% do detector, possuindo calorimetria e a parte central do detector. A figura 3.19 ilustra este protótipo. Várias tomadas de dados empregando múons cósmicos são realizadas com este protótipo com vista à realização de testes de operação do detector. O protótipo permite ao experimento MINERνA verificar a operação e funcionamento do detector antes da montagem final do detector completo com testes do *hardware*, DAQ e reconstrução de eventos. O trabalho desenvolvido por mim e apresentado nesta dissertação foi testado neste protótipo que é descrito no capítulo 6.

3.11.5 Fase de testes

A fase de testes do experimento MINERνA começou em outubro de 2009 e estende-se até o segundo semestre de 2009.O protótipo de rastreamento e o detector de prova usarão, cada um, 100 MAPMT implicando em eventos de cerca de 200 kB. A taxa de eventos para o detector de prova é de cerca de 10 Hz (muito maior que a taxa de eventos do detector completo que será da orde de 0,5 Hz). Na fase de testes coletaremos cerca de 250 GB por semana o que requererá um espaço em disco de 10 TB até o final de 2009[6].

3.11.6 O detector completo

O experimento completo emprega cerca de 500 MAPMT. Um evento sem supressão de zeros será da ordem de 1 MB. A maioria dos eventos, entretanto, será obtido com supressão de zeros e serão da ordem de 100 kB ocorrendo a cada 2 segundos. A taxa de eventos será, então entre 5 GB e 10 GB por dia correspondendo a algo entre 1,5 TB e 3 TB por ano. Em quatro anos

[6]Convém observar que isto é a quantidade que o D0/CDF escrevem por hora

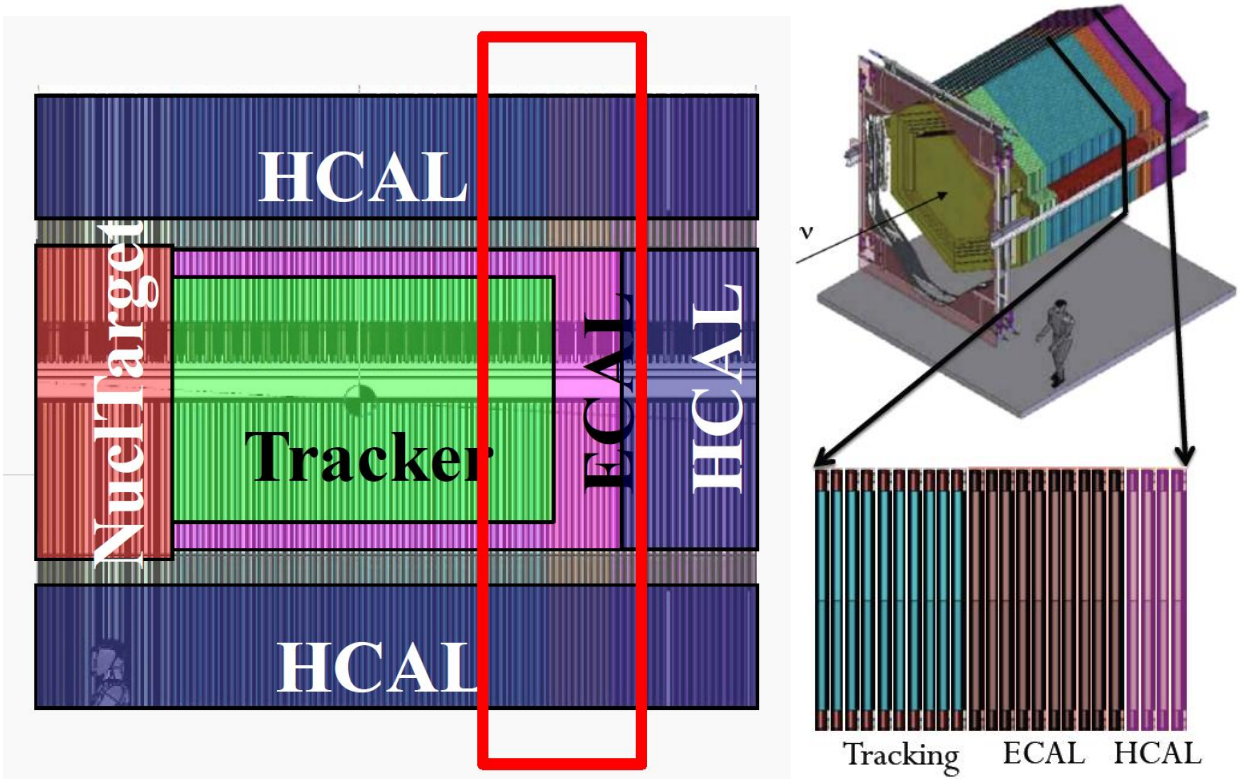

Figura 3.19: Esquema do detector de do MINERνA mostrando a parte que corresponte ao protótipo de rastreamento

de tomada de dados o MINERνA acumulará cerca de 6 TB a 12 TB de dados. Tudo isto será mantido em discos rígidos.

O experimento MINERνA trabalhará em associação com o experimento MINOS de forma a ter acesso a alguns de seus dados que serão empregados na detecção de múons.

3.12 Perspectivas do experimento MINERνA

Na faixa de 1 a 10 GeV dados relevantes vem de experimentos com câmaras de bolhas entre 1960 e 1980. A câmara de bolhas Gargamelle do Argonne, de 12 pés, a câmara de 7 pés de Brookhaven, BEBC no CERN, SKAT em Serpukov e a câmara de 15 pés do FERMILAB estudaram interações de neutrinos e anti-neutrinos com núcleos.

Apesar da baixa estatística, a excelente resolução de imagem das câmaras de bolhas tornou possível a realização de uma grande gama de experimentos. Os dados dos experimentos com câmara de bolhas, além de poucos, não cobrem todo o espectro de energias dos neutrinos, alvos nucleares e tipos de neutrinos ($\nu/\overline{\nu}$) necessário para um entendimento completo das interações de neutrinos. A tabela 3.3 resume a situação experimental.

O detector MINERνA será completado no segundo semestre de 2009 e a tomada de

Experimento	Ano	Reação	Medida	Eventos	Referência
Gargamelle	1977	$\nu/\overline{\nu}$-propano/freon	semi-inclusiva	ν:1.061	[35]
	1977	$\nu/\overline{\nu}$-propano/freon	produção de π	$\overline{\nu}$:1.200	
Gargamelle	1978	$\overline{\nu}$-propano/freon	$\overline{\nu}(\pi^0)$	139	[36]
	1978	$\overline{\nu}$-propano/freon	$\overline{\nu}(\pi^-)$	73	
Gargamelle	1978	ν-propano/freon	$\nu p \rightarrow \nu p \pi^0$	240	[37]
	1978	ν-propano/freon	$\nu p \rightarrow \nu p \pi^+$	104	
	1978	ν-propano/freon	$\nu p \rightarrow \nu n \pi^0$	31	
	1978	ν-propano/freon	$\nu p \rightarrow \nu n \pi^+$	94	
Gargamelle	1979	$\nu/\overline{\nu}$-propano/freon	$\nu(1\pi^0)$	178	[38]
	1979	$\nu/\overline{\nu}$-propano/freon	$\overline{\nu}(1\pi^0)$	139	
BNL-Counter	1977	$\nu/\overline{\nu}$-Al/C	$\nu(\pi^0)$	204	[39]
	1977	$\nu/\overline{\nu}$-Al/C	$\overline{\nu}(\pi^0)$	22	
ANL-12	1974	ν-D_2/ν-H_2	$\nu p \rightarrow \nu n \pi^+$	8	[40]
	1974	ν-D_2/ν-H_2	$\nu p \rightarrow \nu p \pi^0$	18	
ANL-12	1980	ν-D_2	$\nu n \rightarrow \nu p \pi^-$	?	[41]
ANL-12	1981	ν-D_2	$\nu n \rightarrow \nu p \pi^-$	?	[42]
	1981	ν-D_2	$\nu p \rightarrow \nu p \pi^0$	8	
	1981	ν-D_2	$\nu p \rightarrow \nu p \pi^+$	22	
BNL-7	1981	ν-D_2	$\nu n \rightarrow \nu p \pi^-$	200	[43]

Tabela 3.3: Medidas atuais de corrente neutra.

dados deverá começar início de 2010 com duração prevista de quatro anos. Neste período o experimento deverá obter cerca de 14,5 milhões de eventos de corrente carregada (CC). A tabela 3.4 apresenta o número esperado de eventos em diversos tópicos.

Prcoesso	Total	CC/t	CN/t
Quase-elástico	0,8 M	0,27 M	0,09 M
Produção ressonante	1,7 M	0,53 M	0,16 M
Transição: Ressonância para DIS	2,1 M	0,67 M	0,21 M
DIS, Funções de Estrutura e PDF em alto x	4,3 M	1,37 M	0,4 M
Produção coerente de π	0,13 M	0,03 M	0,04 M
Produção de hadrónes charmosos e estranhos	>240 k		
Distribução de Parton generalizada	na ordem de 10 k eventos		
Efeitos nucleares	He: 0,6 M, C: 0,4 M, Fe: 2,0 M e Pb: 2,5 M		

Tabela 3.4: Número de eventos em diferentes tópicos (valores aproximados)

O experimento MINERνA usará, de preferência, a configuracao LE do feixe NuMI por cerca de um ano junto com o experimento MINOS e depois usará a configaração ME junto com o experimento NOνA.

O conteúdo de ν_e no feixe NuMI é estimado ser da ordem de apenas 1% do fluxo. Uma importante função do MINERνA será fornecer uma medida do fluxo e espectro de energia de ν_e no feixe NuMI mais acurada do que a que é possível de ser obtida pelo experimento MINOS.

O MINERνA permitirá medidas mais precisas da secção de choque em baixas energias. Para calcular a secção de choque total temos que considerar as contribuções; $\sigma_{TOT} = \sigma_{QE} + \sigma_{RES} + \sigma_{DIS}$ onde as reações que contribuem em cada secção de choque são :

σ_{QE}: quase-elástico $\rightarrow \nu(\overline{\nu})n(p) \rightarrow \mu^-(\mu^+)p(n)$

σ_{RES}: ressonância $\rightarrow \nu N \rightarrow \mu N^*$

σ_{DIS}: espalhamento profundamente inelástico $\rightarrow \nu N \rightarrow \mu X$

Com os eventos quase elásticos tem-se a perspectiva de calcular, com maior precisão , $\sigma(E_\nu)$ e $d\sigma/dQ^2$, que são importantes para o estudo da oscilação de neutrinos, a determinação do fator forma do vetor axial F_A particularmente em altas Q^2, e o estudo do espalhamento próton intra nuclear e a dependência com o número de massa "A" (alvos de C, Fe, e Pb).

A alta granularidade do detector e sua capacidade de rastreamento permite a identificação de eventos com diversas topologias. A figura 3.20 ilustra a capacidade de identificação de eventos pelo detector. Nesta figura observamos três topologias diferentes e como elas se diferenciam no detector.

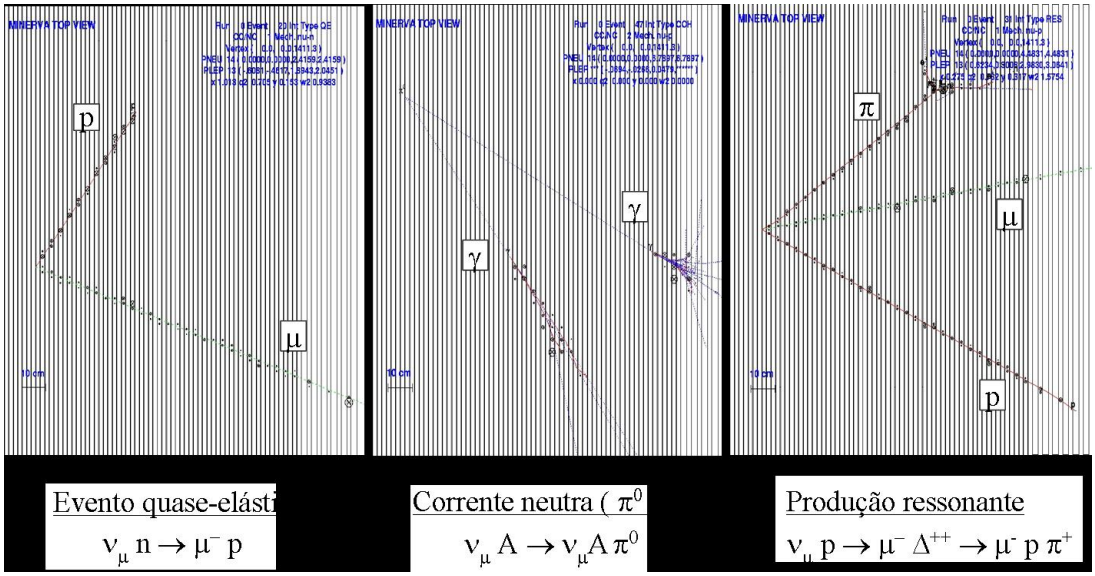

Figura 3.20: Eventos com topologias diferentes podem ser distingüidos graças à capacidade de reconstrução do detector do MINERνA .

Um experimento de espalhamento de neutrinos no feixe NuMI é uma oportunidade excepcional para o estudo de vários tópicos de física com precisão até agora não obtidos. Muitos destes tópicos não foram estudados de forma sistemática e outros apresentam poucos resultados com grandes erros estatísticos e sistemáticos. Alguns dos assuntos possíveis de serem estudados pelo experimento MINERνA são:

- Medida precisa da secção de choque quase elástica neutrino-nucleon, incluindo a dependência em E_ν e em q^2.
- Determinação da secção de choque de produção de píons (simples e dupla) para interações em corrente carregada e corrente neutra.
- Medida precisa da produção coerente de pions com particular atenção à dependência com o número atômico A do núcleo.
- Exame dos efeitos nucleares nas interações de neutrinos.
- Estudos da física nuclear para a qual reações de neutrinos fornecem informações complementares aos estudos realizados na mesma faixa cinemática.

Estes ítens são, por si só, importante tópicos de pesquisas. Além disto, são essenciais para a minimização dos erros sitemático nos experimentos de oscilação de neutrinos.

3.12.1 Espalhamento quase-elástico

Espalhamento quase-elástico domina as interações ν-A na região de $E_\nu > 2$ GeV. A medida precisa da seção de choque para esta reação, incluindo a dependência com a energia e com a variação do alvo nuclear, é fundamental para os experimentos sobre oscilação de neutrinos.

O gráfico superior da figura 3.21 mostra as medidas esperadas pelo MINERνA . O gráfico da parte inferior sumariza o conhecimento atual da seção de choque quase-elástica. Este gráfico mostra que as medidas atuais tem erros muito grandes em toda a faixa de energias do MINERνA .

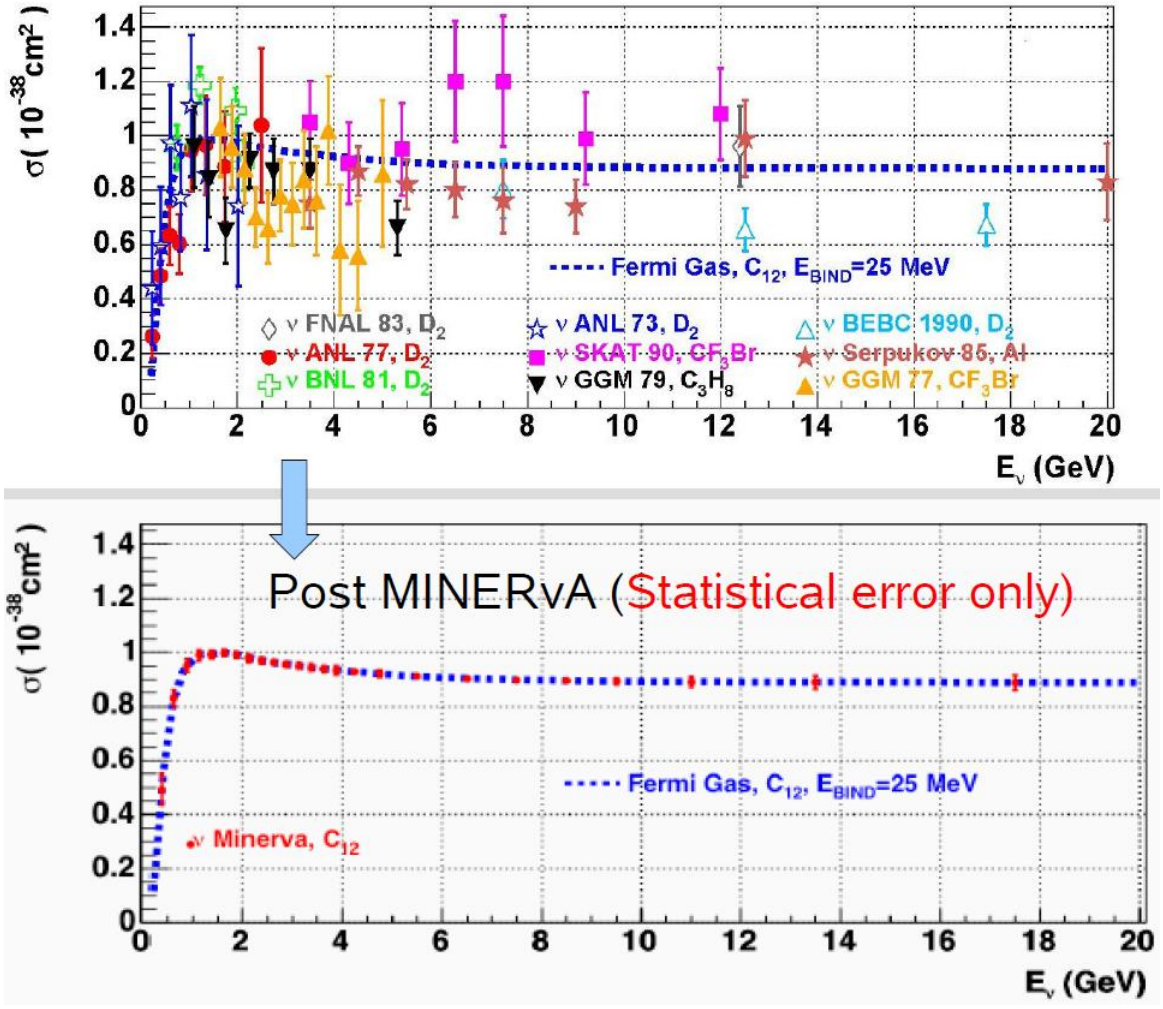

Figura 3.21: superior:Previsão dos resultados de medida da seção de choque quase-elástica esperada pelo MINERνA (apenas erros estatísticos). inferior: medidas atuais da seção de choque quase-elástica. A linha pontilhada em azul emprega um modelo de gás de Fermi.

3.12.2 Produção ressonante de píons

A grande estatística, variedade de alvos nucleraes, baixo patamar de detecção e excelente sistema de rastreamento e calorimetria permitirão que o experimento MINERνA faça melhorias significativas nas medidas da produção ressonante de píons em interações de neutrinos. Estas medidas imporão limites às seções de choque, seções de choque relativas, à forma das distribuições de Q^2 e permitirão as primeiras comparações diretas das interações de neutrinos em alvos diferentes.

3.12.3 Produção coerente de píons

As altas taxas do expermineto MINERνA , os diversos alvos nucleares, a alta granularidade do detector, a grande capacidade de reconstrução de trajetórias e a boa calorimetria do detector tornam possíveis o estudo de espalhamento coerente neutrino-núcleo em corrente carregada e em corrente neutra com precisão até então inatingível.

A figura 3.22 mostra os resultados atuais para produção coerente de píons junto com os resultados esperados pelo MINERνA .

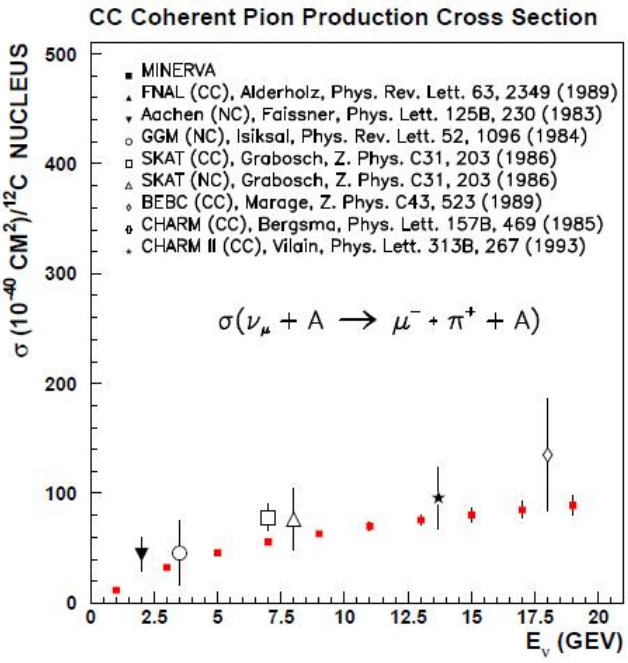

Figura 3.22: Comparação entre os resultados existentes para produção coerente de píons e os resultados esperados pelo experimento MINERνA .

3.12.4 Produção de partículas estranhas e charmosas

O experimento MINERνA realizará os primeiros estudos com alta estatística da produção exclusiva de partículas estranhas por neutrinos. A tabela 3.5 sumariza o tamanho da amostra esperada em diversos canais durante 4 anos de tomada de dados. As seções de choque obtidas terão impacto em outras áreas da física de partículas. O programa de física do MINERνA inclui, também, a busca de novos processos como reações de mudança de estranheza em reações de corrente neutra e ressonâncias bariônicas incomuns. Uma tomada de dados prolongada do feixe NuMi com antineutrinos facilitará estudos de produção de híperons Λ, Σ e Y^* além de permitir novas medidas dos elementos da matrix CKM.

Tipo de reação	Canal exclusivo	N° de eventos
$\Delta S = 0$ CC	$\nu_\mu n \rightarrow \mu^- K^+ \Lambda^0$	23.100
	$\nu_\mu n \rightarrow \mu^- \pi^0 K^+ \Lambda^0$	20.400
	$\nu_\mu n \rightarrow \mu^- \pi^+ K^0 \Lambda^0$	13.800
	$\nu_\mu n \rightarrow \mu^- K^- K^+ p$	11.200
	$\nu_\mu p \rightarrow \mu^- K^0 K^+ \pi^0 p$	3.300
$\Delta S = 1$ CC	$\nu_\mu p \rightarrow \mu^- K^+ p$	34.900
	$\nu_\mu n \rightarrow \mu^- K^0 p$	5.200
	$\nu_\mu n \rightarrow \mu^- \pi^+ K^0$	4.600
$\Delta S = 0$ NC	$\nu_\mu p \rightarrow \nu K^+ \Lambda^0$	7.900
	$\nu_\mu n \rightarrow \nu K^0 \Lambda^0$	2.400
	$\nu_\mu n \rightarrow \nu K^0 \Lambda^0$	6.100

Tabela 3.5: Amostras esperadas pelo experimento MINERνA para diversos eventos de produção exclusiva de partículas estranha durante uma tomada de dados de 4 anos

Um maior detalhamento pode ser encontrado em [34]

Nesta dissertação apresentamos trabalho realizado no desenvolvimento do processamento e monitoramento de dados em tempo real do experimento MINERνA , descritos nos capítulos 4, 5 e 6

Capítulo 4

Software do MINERνA

4.1 Introdução

Software e recursos computacionais são partes fundamentais do MINERνA e prestam-se, principalmente, a três tarefas principais:

1. Armazenagem de dados durante a tomada de dados.
2. Processamento de dados e simulação.
3. Analise dos dados.

Uma simulação detalhada é necessária para que se tenha uma compreensão do detector tão completa quanto possível. Seleção de eventos e reconstrução incorporam-se à simulação para melhorar a nossa compreensão da resposta do detector e a análize dos dados.

Os seguintes s ão alguns dos *software* que precisam ser desenvolvidos ou implementados para o MINERνA:

1. Geração de eventos de neutrinos, incluindo efeitos nucleares em interações neutrino-nucleon;
2. Descrição acurada do detector do MINERνA.
3. Códigos para calibração e alinhamento do detector;
4. Programas para seleção e reconstrução de eventos;
5. Software para a análize e visualização de eventos.

O MINERνA utiliza uma combinação de software já existentes com novos pacotes. Esta abordagem reduz o tempo necessário para realização das tarefas em software. O desenvolvimento dos pacotes envolve o uso de técnicas de programação orientada por objetos, sendo todos os programas desenvolvidos em C++.

O *software* do MINERνA consiste de todos programas desenvolvidos para operação, simulação, reconstrução, análise e visualização dos dados do experimento. O MINERνA tem programas para operação tanto *online* quanto *offline*, muitos deles desenvolvidos fora da colaboração mas que servem às nossas necessidades com pouca ou nenhuma alteração. Os objetivos do *software* do MINERνA são:

- Criar um sistema portável, que possa ser mantido durante a vida útil do experimento, acessível a não especialistas e que permita o experimento alcançar seus objetivos em física. O *software* do MINERνA é baseado no *software* do LHCb.
- Empregar tantos componentes de software já existentes quanto possível.

4.2 Desenvolvimento e Estrutura

Diversos ambientes são empregados pelo MINERνA para desenvolvimento de seu *software*. Entre os quais mencionamos:

- CVS (Concurrent Versioning System): um sistema largamente empregado para gerenciamento de códigos fonte.
- C++: linguagem de programação mais empregada em física de altas energias.
- CMT: pacote de gerenciamento e configuração de sistema que é usado por um grande número de experimentos como ATLAS, LHCb e GLAST. CMT permite que o mesmo código base seja construído e usado em várias plataformas. Ele é o encarregado da compilação dos códigos fonte e da definição das dependências entre pacotes. Mais informações sobre este pacote são encontradas em [44] e detalhes sobre a sua utilização são encontradas no manual do CMT encontrado em [45]
- CLHEP, ROOT, SEAL, POOL, AIDA e outras: bibliotecas e ferramentas externas que não são específicas de nenhum experimento. São gerenciadas, mantidas e atualizadas pelo projeto LCG, do CERN. Todas as edições de nossos *software* são referenciadas a versões específicas do LCG. Isto assegura compatibilidade de todas as ferramentas envolvidas. Mais informação pode-se encontrar na referença [46].

- GAUDI: arcabouço desenvolvido para esperimentos em física experimental de altas energias. GAUDI suporta aplicações que processam eventos que rodem em diferentes ambientes, desde aplicações em tempo real até análize de dados. GAUDI comporta uma grande funcionalidade (como o preenchimento de histogramas ou n-uplas) mas não realiza nenhuma tarefa específica de física como simulação ou visualização. GAUDI depende de um subconjunto de bibliotecas do LCG.

Diversas particularidades do MINERνA requerem algumas modificações no LCG, GAUDI e nas diferentes camadas do software do LHCb. Nestas situações o MINERνA cria uma copia dos pacotes envolvidos, preservando os originais. Quando as alterações são extensões ou correções que sejam compatíveis com outros experimentos o MINERνA solicita que os mantenedores dos pacotes incorporem as alterações nas novas versões. Isto minimiza o número de códigos que sejam específicos do MINERνA.

A estrutura geral do *software* do MINERνA, assim como a do LHCb, está baseado inteiramente no CMT que é o sistema de compilação e ligação de bibliotecas e aplicativos. O CMT permite ao MINERνA utilizar e depender do *software* do LHCb de uma maneira natural. A figura 4.1 mostra a dependência do software específico do MINERνA com o *software* do LHCb e outros projetos padrões que são utilizados na comunidade de experimentos de física de neutrinos (GENIE). O *software* externo usado pelo MINERνA que não foi feito em CMT (GENIE e PYTHIA principalmente) precisaram ser adaptados para.

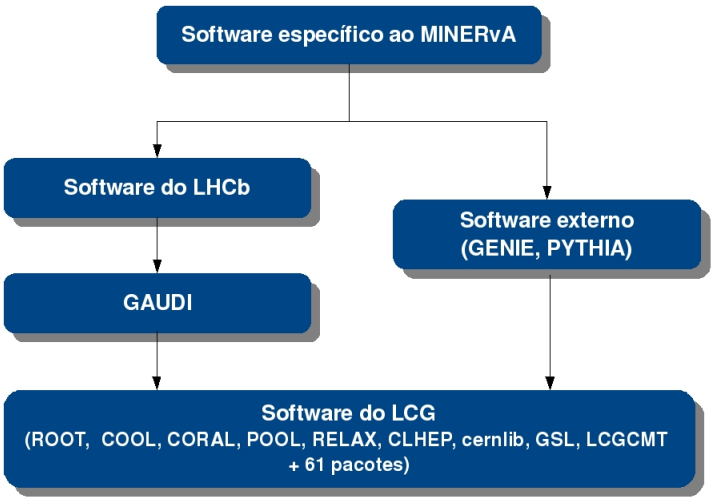

Figura 4.1: Hierarquia na estrutura do *software* utilizado pelo experimento MINERνA em fevereiro de 2009. A dependêcia do *software* especifico do MINERνA com outros projetos desenvolvidos fora da colaboração MINERνA é posível devido ao CMT

O CMT divide todo o *software* utilizado no experimento MINERνA em vários pacotes CMT[1] agrupados em projetos CMT[2] onde cada pacote é uma coleção de classes relacionadas. Desta maneira, todo o *software* utilizado no MINERνA lida apenas com pacotes e não com classes ou subrotinas individuais.

Cada módulo mostrado na figura 4.1 representa um conjunto de um ou mais projetos desenvolvidos independentemente. Nesta hierarquia, o *software* específico do MINERνA encontra-se no topo e pode-se notar sua dependência direta com o *software* do LHCb que, por sua vez, depende do projeto GAUDI. Ambos projetos são desenvolvidos e mantidos fora da colaboração MINERνA. O projeto GAUDI define o arcabouço no cual todo o *software* do LHCb e o *software* específico do MINERνA são desenvolvidos. Uma apresentação geral do arcabouço GAUDI é dada a seguir. Informações mais detalhadas são encontradas nas referências [47] e [48].

4.3 O Arcabouço GAUDI

Em desenvolvimento de software, um arcabouço (*framework*) é um esqueleto de um programa no qual os programadores podem inserir seus códigos e que fornece a maior parte da funcionalide comum.

GAUDI[3] é um arcabouço orientado a objetos que pode ser usado para o desenvolvimento de todas as aplicações para processamento de dados em diversos ambientes. Isto inclui processamento de dados obtidos diretamente do sistema *online*, a simulação de eventos que roda *offline* e a visualização de eventos que ocorre de forma interativa. O uso de um mesmo arcabouço em todas as aplicações assegura a integridade do projeto de *software*.

O arcabouço GAUDI permite uma separação entre algorítmos e dados bem definida. Na prática isto significa que os algorítmos são objetos definidos mediante classes do tipo *GaudiAlgorithm* enquanto que os dados são objetos, chamados objetos de dados, definidos mediante classes do tipo *DataObject* ou *DataContainer*. A diferença entre um objeto de dados baseado na classe *DataObject* e o baseado na classe *DataContainer* é que este último, mais frecuentemente usado, é um conjunto de objetos tipo *DataObject*.

Objetos de dados são usados para descrever toda a informação , processada ou não , obtida pela leitura de todo o detector ou pela simulação do mesmo. Por outro lado, os algorítmos são os lugares previstos no arcabouço para os códigos do usuário final. Eles processam objetos de

[1]Um pacote CMT constitui a menor entidade que pode ser construída em diferentes versções.

[2]Daqui para frente nos referiremos aos projetos e pacotes CMT simplesmente como projetos e pacotes respectivamente.

[3]GAUDI foi desenvolvido no contexto do experimento LHCb de forma a ser facilmente customizado e adaptado a tarefas diferentes. Pôde, então, ser facilmente adaptado ao MINERνA.

dados de algum tipo e produzem novos objetos de dados de um tipo diferente.

GAUDI permite que vários algorítmos rodem num mesmo aplicativo. O fluxo de dados entre algorítmos se dá via a Armazenagem Transiente de Eventos (TES). A TES atua como um "quadro-negro"entre os algorítmos. A figura 4.2 ilustra como um fluxo de dados em particular pode ser alcançado com o emprego da TES e pelo sequenciamento adequado de algorítmos. Os objetos de dados na TES são organizados em uma estrutura em forma de "árvore", similar ao sistema de arquivos do Unix como é mostrado na figura 4.2. Cada uma destas "pastas"representam dados correspondentes a diferentes partes do detector ou em diferentes estágios de processamento. Após processamento, todos os objetos de dados presentes na TES podem ser facilmente éscritos em arquivo sem que seja necessário qualquer código adicional. De uma maneira similar, objetos de dados num arquivo podem ser lidos por outro aplicativo e popular sua TES. Algorítmos vêm apenas objetos de dados na representação transiente e, consequentemente, são blindados da tecnologia empregada para seu armazenagem persistente.

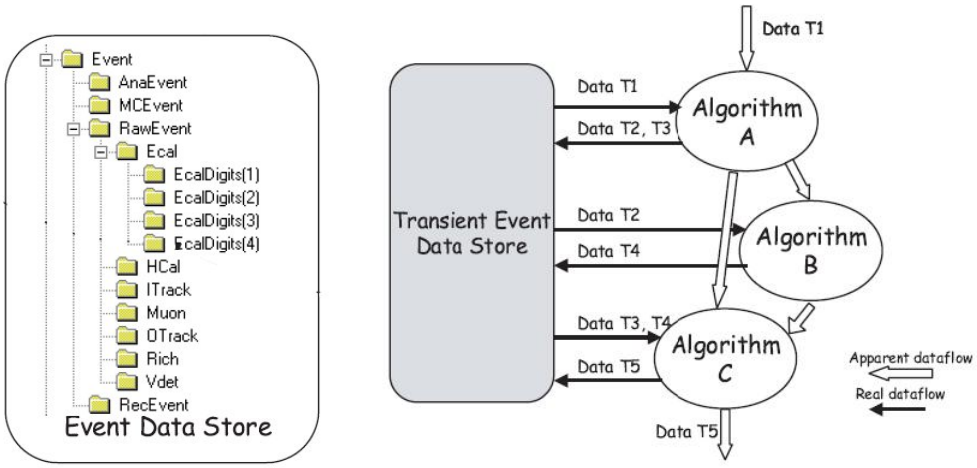

Figura 4.2: Exemplo da estrutura da Armazenagem Transiente de Eventos e de como ela é usada pelos algorítmos

Serviços GAUDI fornecem a funcionalidade do arcabouço. Esta categoria de componentes é disponível globalmente e deve fornecer todos os serviços necessários aos algorítmos. Esta abordagem libera o usuário (que faz os algorítmos) de ter que desenvolver as tarefas rotineiras de *software* que são tipicamente necessárias em aplicações de processamento de dados físicos. Um exemplo é o serviço encarregado de escrever os objetos na TES em um aequivo em disco.

No topo da hierarquia de algorítmos encontra-se o gerente de aplicações, que sabe em quais algorítmos iniciar e quando chamá-los. Os diferentes aplicativos para processamento de dados são feitos rodando o gerente de aplicações customizado mediante um arquivo de opções (em geral um arquivo com extenção .opts) onde se faz a escolha apropriada do conjunto de algorítmos a

ser executado. Isto torna possível construir aplicativos pela simples seleção dos componentes mais adequados à realização das diferentes tarefas.

4.4 Os projetos específicos do MINERνA

O *software* específico do MINERνA consiste de vários pacotes agrupados em projetos. Dentro destes pacotes são definidos algorítmos, objetos, ferramentas, serviços, interfaces GAUDI assim como classes específicas que o MINERνA desenvolve para realizar diferentes tarefas.

Ao iniciar meu trabalho no experimento MINERνA, em maio de 2008, a estrutura dos projetos específicos do MINERνA era como mostrado na figura 4.3 e consistia de 7 projetos com aproximadamente 30 pacotes.

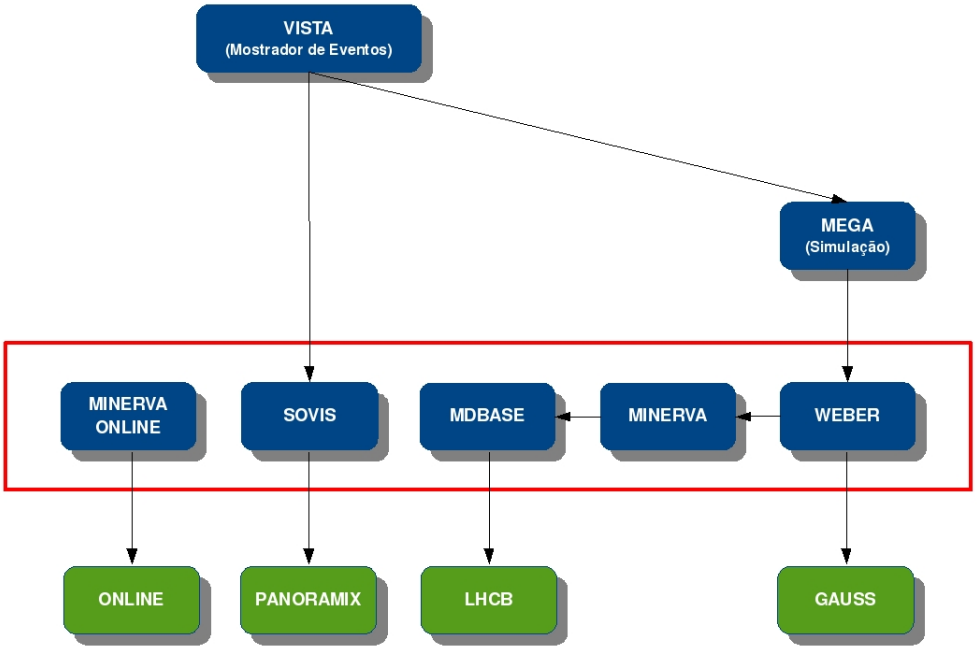

Figura 4.3: Detalhes da dependência do *software* do MINERνA v3r1 com o *software* do LHCb em maio 2009. Azul: *software* específico do MINERνA. Verde: *software* do LHCb.

Só os projetos do *software* do LHCb que estão diretamente ligados com os projetos do *software* do MINERνA são mostrados na figura 4.3. Todos eles dependem, principalmente, do projeto GAUDI e de projetos no *software* do LCG assim como de alguns outros projetos dentro do mesmo *software* do LHCb. Estas dependências são muito complicadas e extensas e nem todos os projetos no *software* do LHCb são usados pelo MINERνA. Uma descrição deles não será apresentada aqui mas pode ser encontrada nas referências [49] e [50].

Apresentamos a seguir uma breve descrição dos projetos do LHCb dos quais o *software* específico ao Minerνa depende diretamente.

- LHCB, projeto de propósito geral para códigos específicos do LHCb.
- GAUSS, projeto que define as aplicações dedicadas a la geracão de colisiones e simulação do detector do LHCb.
- ONLINE, projeto que define os componentes e ferramentas usadas na adquisição e processamento de dados em tempo real no LHCb.
- PANORAMIX, projeto que define a aplicação encarregada da visualização de eventos, simulados ou reais, no detector LHCb.

A primeira camada de projetos acima do *software* do LHCb, que é mostrada na figura 4.3 dentro de uma caixa vermelha, serve como uma interface formada por diversos projetos contendo aplicativos e bibliotecas. São eles:

- MINERVA, projeto de propósito geral para códigos específicos do MINERνA.
- MDBASE, descrição do detector e dados sobre as condições da tomada de dados.
- WEBER, interface com o projeto de simulação no LHCb, baseado no GEANT4 e vários pacotes especializados disponíveis no LCG.
- SOVIS, interface entre o projeto de visualização do detector e de visualização de eventos do LHCb (PANORAMIX) e o equivalente do MINERνA (VISTA).
- MINERVAONLINE, interface e aplicativos necessários para rodar a aquisição e processamento de dados em um ambiente *online*.

A camada superior contém aplicativos para o usuário final do *software* do MINERνA. Nesta versão , eles são usados apenas na simulação do detector. São eles:

- MEGA, o projeto de simulação do detector do MINERνA e gerador de eventos.
- VISTA, o projeto de visualização do detector do MINERνA e visualizador de eventos.

Ao final de meu trabalho, em fevereiro de 2009, a estrutura de projetos específicos do MINERνA era como mostrada na figura 4.4 consistindo de 11 projetos com aproximadamente 50 pacotes. A lista de pacotes é apresentada no apêndice A.

Nesta versão foi adicionado o projeto EXTERNAL como uma dependência a mais ao *software* do LHCb que é usado, basicamente, para a simulação do eventos. Neste projeto encontra-se pacotes especializados que o MINERνA precisa usar (como o GENIE e o PYTHIA) e que não são disponíveis nem no LHCb nem no LCG.

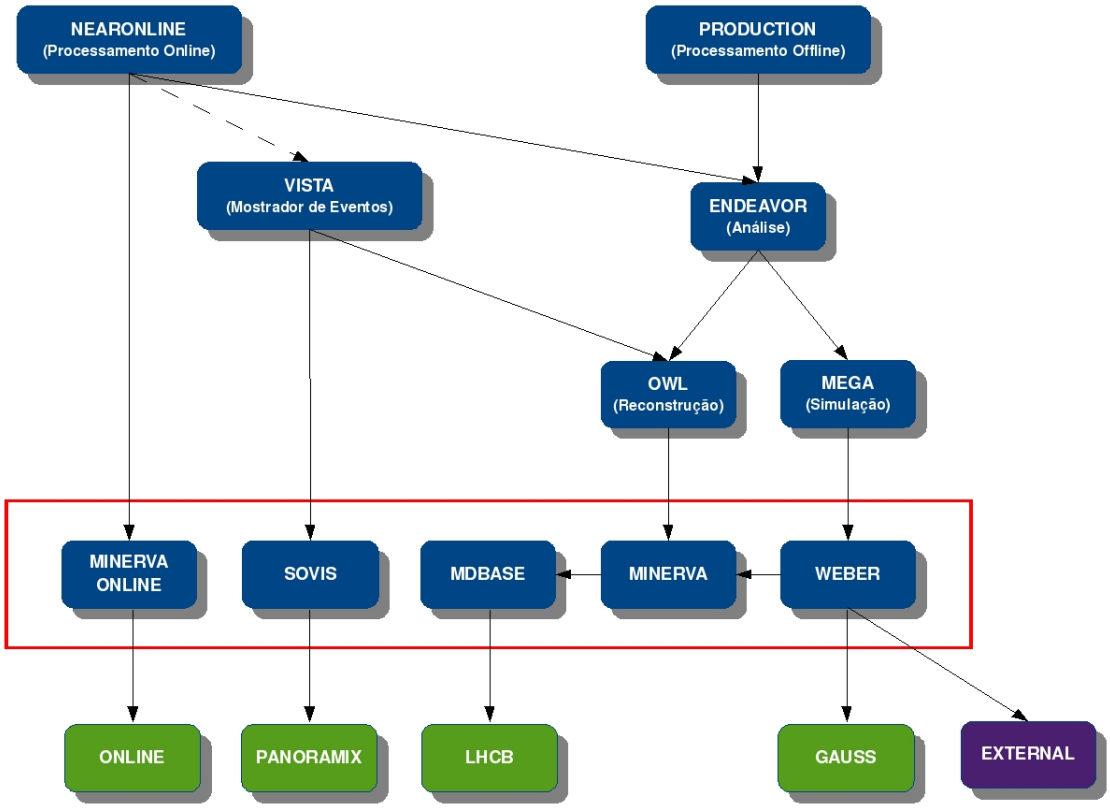

Figura 4.4: Detalhes da dependência do *software* de MINERνA v6r1 com o *software* do LHCb em fevereiro de 2009. Azul: *software* do MINERνA. Verde: *software* do LHCb e Lilás: *software* externo. A seta tracejada entre NEARONLINE e VISTA significa que esta dependência ainda não está totalmente implementada.

A primeira camada de projetos acima do *software* do LHCb não mudou em estrutura mas sim em conteúdo. Agora ela serve como uma interface formada por diversos projetos que não comtêm aplicativos, somente bibliotecas. São eles:

- MINERVA, projeto de propósito geral para códigos específicos do MINERνA. O processamento de dados é feito em camadas superiores.
- MDBASE, descrição do detector e dados sobre as condições da tomada de dados.
- WEBER, interface para o projeto de simulação no LHCb, baseado no GEANT4 e vários pacotes especializados disponíveis no LCG.
- SOVIS, interface entre o projeto de visualização do detector e de visualização de eventos do LHCb (PANORAMIX) e o equivalente do MINERνA (VISTA).
- MINERVAONLINE, interface e aplicativos necessários para rodar a adquisição de dados em um ambiente *online*. O processamento *online* é feito agora por projetos em camadas

superiores.

Os seguintes projetos são uma nova camada encarregada de definir algorítmos e ferramentas de GAUDI que são usadas pelos aplicativos nas camadas superiores. São eles:

- OWL, o projeto de reconstrução e calibração do MINERνA. Contém os algorítmos utilizados no processamento de dados.
- MEGA, o projeto de simulação do detector do MINERνA e gerador de eventos.
- ENDEAVOR, o projeto de análise de dados do MINERνA.

A camada superior contém aplicativos para o usuario final do *software* do MINERνA. São eles:

- VISTA, o projeto de visualização do detector do MINERνA e visualização de eventos.
- NEARONLINE, o projeto de monitoramento e processamento de dados *online*. Foi resultado de meu trabalho e é apresentado em detalhes no capítulo 5.
- PRODUCTION, o projeto de processamento de dados offline do MINERνA.

Alguns pacotes desta última camada de projetos não contêm nenhum código (por exemplo, DaqRecv no NEARONLINE ou SystemTest no PRODUCTION). O conteúdo destes pacotes é majoritariamente *scripts* ou arquivos de configuracção que o usuário final pode editar, o que lhe permite utilizar todos os Algorítmos, Ferramentas, Serviços GAUDI definidos nos pacotes dos projetos nas camadas inferiores.

4.5 O Modelo de Evento

Ao conjunto de dados reais ou simulados, obtidos com o sistema de aquisição do detector ou o *software* de simulação respectivamente, que representam a leitura completa do detector chama-se de Evento. Os Eventos completos são descritos por objetos de dados. Para o processamento dos Eventos é preciso ter uma descrição , específica de cada experimento, de como eles evoluem a medida que passam pelos algorítmos de processamento. A este conjunto de Eventos e Eventos processados chama-se de Modelo de Evento ou *Event Model*. Ele constitui a base e é a parte mais importante do *software* de processamento empregado no MINERνA. Estes objetos são apresentados na figura 4.5 onde se mostra como eles evoluem ao longo do processamento.

O Modelo de Evento consiste do conjunto de objetos (tipo *DataObjects* ou *DataContainer*) que descreve um Evento nos diferentes estágios de processamento. Cada um destes objetos contém informação bem definida e particular a cada estágio do procesamento. São eles:

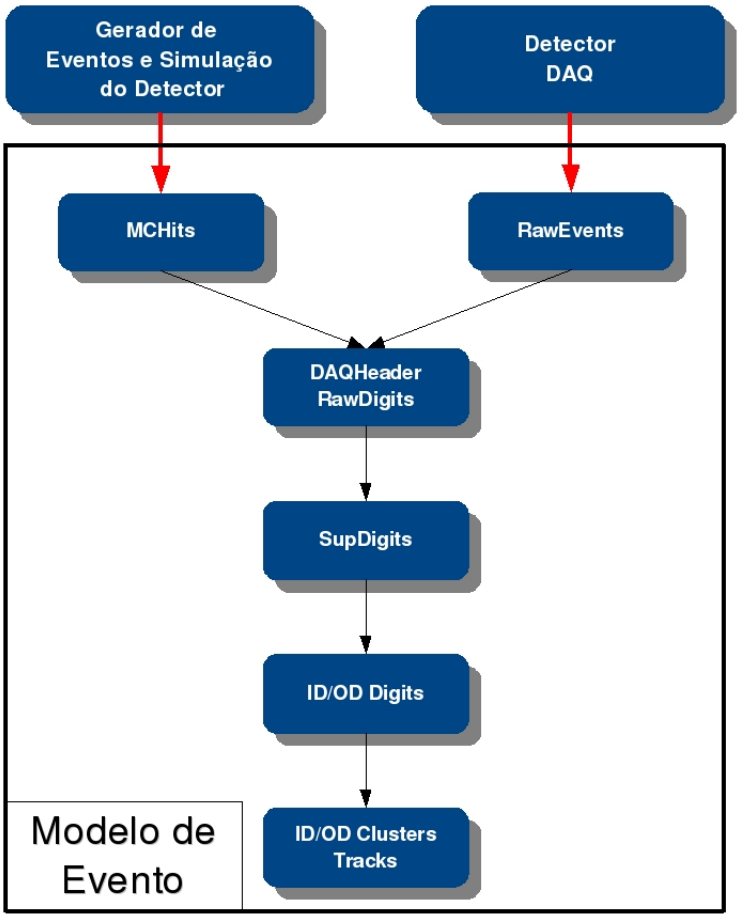

Figura 4.5: Diagrama do Modelo de Evento usado no experimento MINERνA.

- MCHits, são criados pelo *software* de simulação e representam a deposição de energia, a posição e tempo em que a energia foi depositada. Ele contém informação da partícula que depositou a energia.

- RawEvents, são criados pelo *software online* conectado ao detector do MINERνA e cada um representa um único evento. Contém os valores dos sinais lidos de todos os canais de todas as PMT. Esta informação é guardada como um *array* de números inteiros dentro do *RawEvents*. Este objeto é idêntico ao do LHCb.

- DaqHeader e RawDigits, contém a mesma informação que o RawEvent, mas em formato inteligível às pessoas. O DAQHeader contém informação especifica do Evento como: o número do Evento na tomada de dados, o número da tomada de dados, o tipo de Evento, etc. RawDigits é um objeto de dados baseado na classe *DataContainer* que contém objetos *RawDigt*[4]. Cada *RawDigit* contém a informação de amplitude e tempo do sinal em um

[4]Note-se que o nome deste objeto difere de RawDigits apenas por um "s"no final.

canal do detector. Assim, existem tantos objetos *RawDigit* em um objeto *RawDigits* quanto canais no detector.

- SupDigits, é um objeto de dados baseado na classe *DataContainer* que contém apenas os objetos *RawDigit* que representam canais das PMT com sinal de luz.

- IDDigits e ODDigits, são 2 objetos tipo *Container* e contêm objetos *IDDigit* e *ODDigit* respectivamente. Os objetos *IDDigit* e *ODDigit* foram projetados para conter informação calibrada acerca dos canais correspondentes ao *Inner Detector*(ID) e ao *Outer Detector*(OD) respectivamente. Transforma a ID eletrônica de cada elemento dos PMTs (número do elemento, número do PMT, etc) em uma ID do detector (número do plano, número da faixa cintiladora, etc) mediante um mapeamento definido no pacote Det/Plex no projeto MINERVA.

- IDCluster, ODCluster e Tracks, são os objetos reconstruídos representando aglomerados (*clusters*) de digitos e trajetórias completas, respectivamente.

Os objetos pertencentes ao Modelo de Evento são definidos em pacotes especiais dentro do *software* do MINERνA o do LHCb. Estes pacotes devem ser estábeis e construídos de maneira que só mudem minimamente durante o desenvolvimento do *software*. Alterações nestas classes tornam muito difícil manter-se a compatibilidade a medida que o *software* evolui. No caso do MINERνA estes pacotes encontram-se no projeto MINERVA. Observa-se na figura 4.4 que todos os projetos específicos do MINERνA (sem considerar aqueles que servem como interface ao *software* do LHCb) dependem do projeto *MINERVA*. Isto é devido a que ele contém o Modelo de Evento.

4.6 O Processamento de Dados no MINERνA

O procesamento dos eventos é feito de duas maneiras distintas, dependendo se os eventos originam-se em uma simulação ou se eles foram obtidos pelo sistema de aquisição de dados. Ambas as alternativas só se distinguem no primeiro estágio.

Primeiramente, dados da simulação ou do detector precisam ser transformado em objetos do Modelo de Evento. Estes objetos são objetos de dados GAUDI e podem ser escritos na TES e ser processados usando algorítmos GAUDI.

Os dados simulados são gerados pelo gerador de eventos de neutrinos GENIE cuja saída é convertida em *MCHits* por pacotes no projeto *MEGA*. O mesmo projeto tem pacotes que podem realizar a conversão de *MCHits* em *DAQHeader* e *RawDigits*. Esta conversão utiliza informação acerca da geometría, materiais, resposta eletrônica e óptica do detector. Esta

conversão é bastante complexa (envolve vários algorítmos e estágios) e não é discutida nesta tese. O processamento de dados que foi utilizado em meu trabalho é descrito a seguir.

O sistema de adquisição de dados (sistema *online*) que usa os pacotes do projeto MINERVAONLINE lê os dados do detector e converte-os em *RawEvents*. Eles são escritos em disco para armazenagem permanente. Mais informação de como isto é feito encontra-se no anexo C.

O MINERνA emprega uma única máquina dedicada para aquisição de dados que não pode ser utilizada por outras tarefas. Todo o processamento subsequente é realizado *offline* ou quase-*online* em máquinas diferentes. O processamento quase-*online* ́parte importante de meu trabalho e é descrito no capitulo 5. Primeiro descrevemos o procesamento *offline*.

Após a criação de vários arquivos contendo *RawEvents* (e cada um contendo vários Eventos), eles são copiados para outras máquinas. Nestas máquinas os arquivos podem ser lidos por processos GAUDI para continuar o processamento, como ilustrado na figura 4.6.

Com a estrutura de *software* mostrada na figura 4.3 apenas os algorítmos *DecodeRawEvent* e *BasicHistos* eram utilizados.

- DecodeRawEvent, algorítmo cuja função é simplesmente desempacotar os dados contidos no *RawEvents* e criar o *DAQHeader* e *RawDigits*. Este algorítmo pode ser encontrado no pacote Event/MinervaEvent no projeto MINERVA.

- BasicHistos, algorítmo capaz de criar histogramas a partir da informação dos *RawDigits*. Ao final da tomada de dados, é criado um histograma por cada *RawDigit* (resultando ao final em tantos histogramas quanto canais de PMT) e o arquivo resultante é usado para analisar rapidamente a resposta das fotomultiplicadoras. Os resultados são mostrados no capitulo 6. Este algorítmo (como qualquer outro que crie histogramas) podese ser usado para o monitoramento *online*. Mais informação acerca disto é apresentada no capítulo 5.

Com a estrutura apresentada na figura 4.3 cada algorítmo rodava em aplicativos diferentes, um depois de outro. Cada aplicativo lia o arquivo do aplicativo anterior de maneira que ao final geramvam-se apenas 3 arquivos, cada um contendo um objeto do Modelo de Evento (*RawEvents*, *RawDigits* e *ID/OD Digits*). Este procedimento tornava-se muito trabalhoso a medida que aumentava o número de algorítmos. Cada colaborador precisava desenvolver seu próprio aplicativo GAUDI e um conjunto de *scripts* de maneira a automatizar o processo.

Nas versões mais recentes do *software*, como a mostrada na figura 4.4, introduziu-se mais algorítmos e criou-se o projeto PRODUCTION que passou a ocupar-se da automatização do processamento.

Usando PRODUCTION, o usuário executa só um aplicativo capaz de processar os dados a partir dos *RawEvents* até os *Tracks*. O aplicativo cria um arquivo contendo todos os objetos de

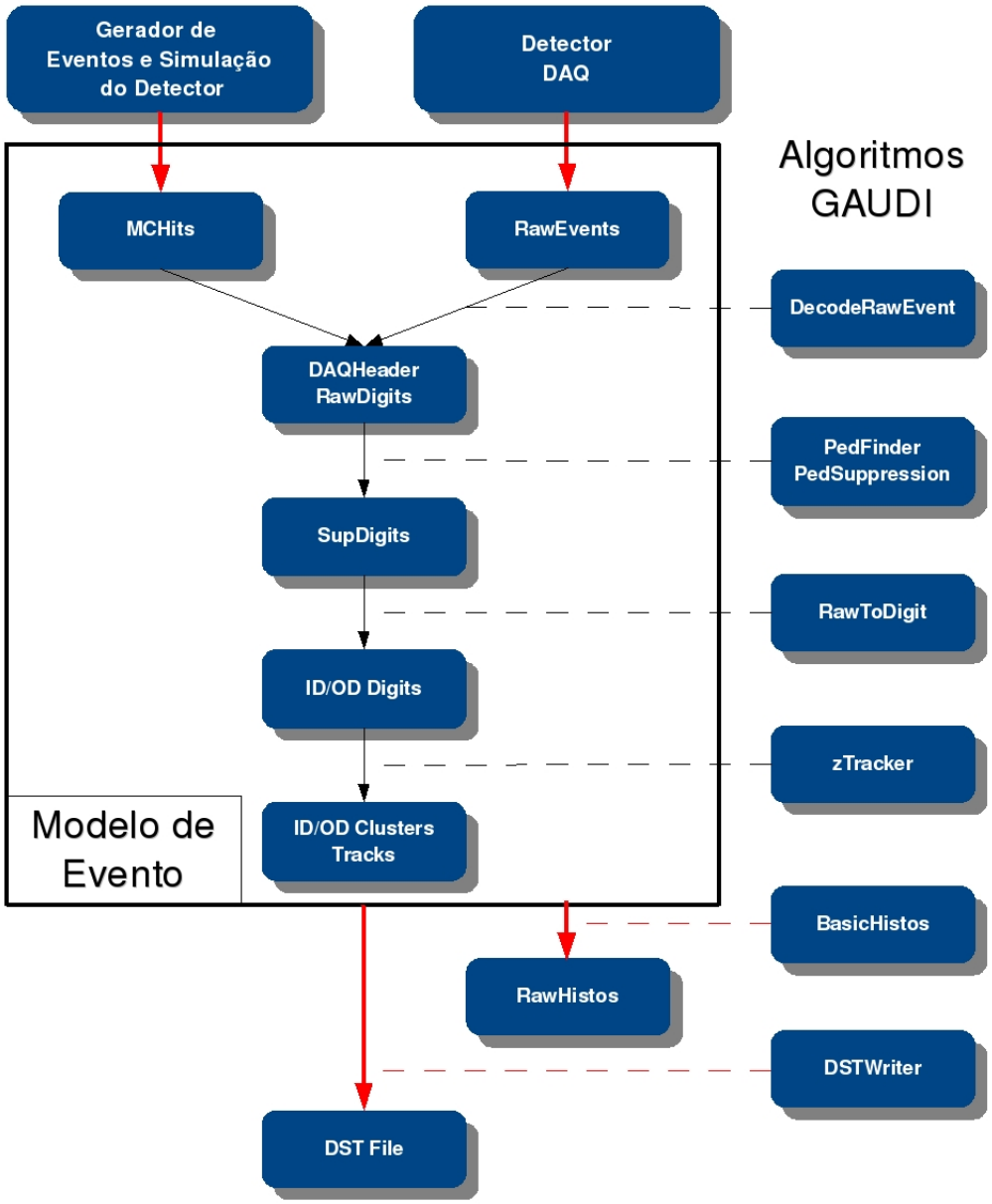

Figura 4.6: Esquema que descreee o processamento de dados no experimento MINERνA. Os objetos do Modelo de Evento são mostrados junto aos algoritmos utilizados entre eles.

dados no Modelo de Evento. Este arquivo só pode ser utilizado dentro do arcabouço GAUDI e, por isto, cria-se, adicionalmente, um arquivo DST (*Data Summary Tape*) contendo toda a informação de cada objeto no Modelo de Evento que foi criado em uma n-upla em formato ROOT ROOT para ser analisado pelos colaboradores. O conteúdo da DST pode ser visto no anexo B.

Capítulo 5

Processamento e Monitoramento em Tempo Real

5.1 Introdução

A qualidade dos dados produzidos por um experimento depende do bom funcionamento do detector durante todo o período de tomada de dados. É por isto que o monitoramento em tempo real dos dados coletados e do detector é tarefa fundamental à operação de um experimento de física de altas energias. Ele permite a identificação de problemas e sua imediata solução sem comprometer uma fração significativa dos dados.

O sistema de aquisição de dados do experimento MINERνA é dividido em três partes.

- O sistema de controle lento, que controla e monitora parâmetros cuja variação é lenta e os escreve em uma base de dados. É também responsável pela configuraçao de toda a eletrônica do detector.

- O sistema de aquisição de dados (DAQ) principal, que lê os dados do detector e os escreve em disco. Este sistema é chamado *online* e esta sendo desenvolvido no projecto MINERVAONLINE no *software* do MINERνA.

- O sistema de processamento de dados que recebe os dados do DAQ e os processa em tempo real. Este sistema é chamado *quase-online* e esta sendo desenvolvido no projecto NEARONLINE no *software* do MINERνA.

O sistema *online* de aquisição de dados (DAQ) emprega uma rede local e segura para conectar-se ao sistema *quase-online* de processamento de dados. O sistema quase-online emprega a rede do FERMILAB para publicar histogramas que resultam do processamento, permitindo o monitoramento remoto de dados em tempo real. Cada um dos sistemas armazena

temporariamente os dados que geram em um disco rígido comum aos dois. Depois de acumular uma certa quantidade de dados, eles são enviados aos servidores dedicados para que sejam armazenados permanentemente. A figura 5.1 ilustra o sistema.

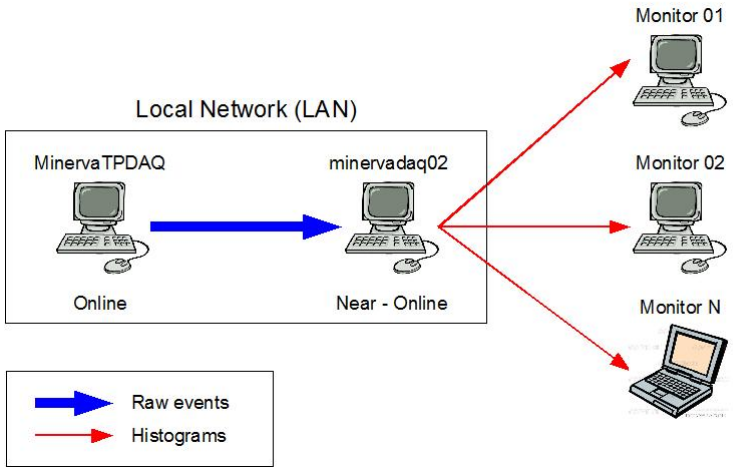

Figura 5.1: Esquema geral do *software* de aquisição de dados do MINERνA.

Os computadores utilizados para o sistema *online* e quase-*online* empregam, cada, dois processadores quad-core INTEL XEON X5472 3.0GHZ 12M 1600MHZ. Uma descrição detalhada do sistema *online* de aquisição de dados (DAQ) e da electrônica envolvida é encontrada na documentação do experimento [51]. No sistema *quase-online* de processamento de dados são executados diversos algorí tmos que processam os dados segundo o Modelo de Evento do MINERνA. Nos diferentes estágios do processamento, algoritmos dedicados ao monitoramento gerão dados estatísticos em forma de histogramas que são enviados a varios computadores para serem exibidos em tempo real durante todo o período de tomada de dados.

Nesta dissertação descrevemos o trabalho realizado em:

- Atualização do sistema *online*.
- Criação , desenvolvimento e instalação do sitema quase-*online* de processamento de dados.
- Desenvolvimento de algorítmos e ferramentas usados pelo sistema quase-*online* que permitem o monitoramento em tempo real do experimento MINERνA

Como em todo o MINERνA os sistemas *online* e quase-*online* empregam o arcabouço GAUDI, descrito no capítulo 3.

Descrevemos, inicialmente, as ferramentas empregadas no desenvolvimento do projeto NEARONLINE (o sistema de processamento de dados em tempo real) e, então, damos detalhes de sua estrutura e funcionalidade. Uma descrição detalhada do uso deste pacote encontra-se no apêndice C.

5.2 Ferramentas utilizadas

Para o desenvolvemento do projeto NEARONLINE foram utlizadas ferramentas já existentes desenvolvidas como pacotes dentro do projecto ONLINE do LHCb. Elas são:

- DIM
- GaudiOnline
- GAUCHO
- HPDMonitor
- Gerente de multi-buffers (Multi-Buffer manager ou MBM)

As ferrramentas GaudiOnline e Gerente de Multi-buffer foram também usadas na implementação do projecto MINERVAONLINE e em sua atualização .

5.2.1 DIM

O DIM (Gerenciamento de Informação Distribuída - *Distributed Information Management*) é um pacote leve de comunicação entre processos baseado no paradígma cliente/servidor. Servidores são definidos como processos que têm alguma informação para publicar (um serviço) e Clientes são os processos que precisam desta informação . DIM é o principal pacote para comunicação de rede no modelo de *software* do LHCb e MINERνA. A figura 5.2 mostra os componentes do DIM e como eles interatuam.

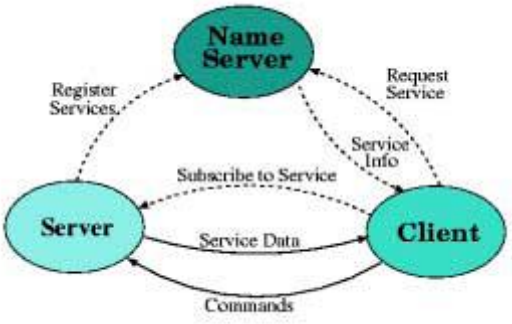

Figura 5.2: Diagrama de fluxo de dados do DIM

Todos os servidores DIM registram-se no Servidor de Nomes (Servidor de Nomes do DIM ou DNS) e publicam todos os seus serviços. O DNS armazena uma lista com todos os servidores que tenham se registrado e os serviços que cada um deles oferece. Quando um cliente DIM é

iniciado ele solicita um serviço ao DNS e recebe a informação da localização do servidor que oferece este serviço. O Cliente, então, registra-se diretamente no servidor e a comunicação tem início. Isto significa:

- O Cliente pode receber informação atualizada em intervalos regulares.
- O Cliente pode receber informação atualizada sempre que ocorrer alguma alteração.
- O Cliente pode enviar comandos ao servidor.

Os três elementos básicos do DIM (mostrados na figura 5.2) podem rodar em máquinas diferentes (ou na mesma máquina para testes) e podemos ter vários deles. Clientes e servidores precisam saber a localização (endereço IP) onde o DNS está rodando. A principal biblioteca do DIM define todas as classes necessárias para a implementação de clientes ou servidores DIM em uma aplicação .

Os serviços publicados pelos servidores podem ser um tipo de dado simple (integer, float, double, short, longlong, char) ou, uma vez que o tamanho seja especificado, um tipo de dado complexo consistindo de um conjunto misto de tipos de dados simples. Há um tipo especial de serviço chamado de comando DIM empregado pelos servidores para receber comandos dos clientes. O nome do serviço é livre (DIM não coloca restrições na forma dos nomes dos serviços) e cabe ao usuário a definição da convenção para os nomes.

Informação adicional acerca do DIM pode ser encontrada na referência [52].

5.2.2 Gaudi Online

Na estrutura do GAUDI algorítmos nunca se comunicam diretamente com a armazenagem permanente de dados, o que a torna adequada para aplicativos *online* onde os dados vêm do DAQ. Apenas o Serviço de Entrada da Armazenagem Transiente, o controle de programa e os componentes de monitoramento precisam ser especializados para se interfacearem com o DAQ e com o sistema de controle, enquanto outros componentes podem ser usados da mesma forma que em aplicativos *offline*. Estas especializações são feitas no pacote GaudiOnline.

Quando um programa GAUDI *online* roda, ele cria um servidor DIM cujo nome é dado pela variável UTGID (User assigned unique Thread Group Identifier). Este nome não é, necessariamente, o nome do processo. Cada um destes servidores DIM publica um serviço de comando (descrito na seção 5.2.1) usando o próprio nome do servidor (UTGID) de forma que eles podem receber comandos (config, start, stop, reset e unload) que controlam o Gaudi-ApplicationManager dentro deles. Eles também publicam dois serviços adicionais chamados UTGID/status e UTGID/fsm_status que fornecem o estado do processo (not_ready, ready,

stopped ou runnning) e o estado da transição entre estados (transition complete, transition failed ou executing transition) respectivamente. A documentação do LHCb refere-se a eles como estados FSM. A figura 5.3 mostra todos estes possíveis estados e transições.

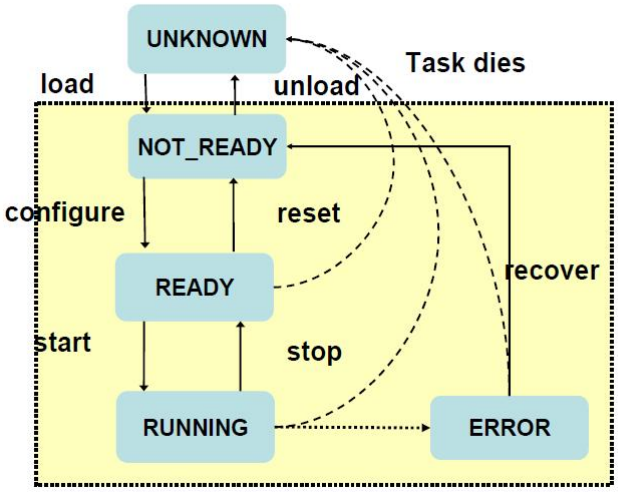

Figura 5.3: Diagrama de estado dos processos rodando como programas GAUDI *online*.

5.2.3 GAUCHO

GAUCHO (GAUdi Component Helping Online) é o pacote que permite que algorítimos em um programa GAUDI *online* publiquem variáveis e histogramas. Ele define o serviço MonitorSvc que é uma implementação da interface GAUDI IMonitorSvc. MonitorSvc torna possível a publicação de variáveis internas (bool, integer, long, double, string, par de doubles e arrays) criadas como serviços DIM.

O MonitorSvc também permite a publicação dinâmica de histogramas encontrados na armazenagem transiente. Isto significa que o usuário não tem que chamar função alguma para publicar histogramas, o MonitorSvc fará isso automaticamente com todos os histogramas criados. Os histogramas são publicados como um serviço DIM e contêm toda a informação como um *array* de inteiros.

O MonitorSvc também inicia um DimPropertyServer. Esta classe é capaz de usar DIM para obter ou fornecer, remotamente, os valores das propriedades de programas GAUDI-*online* (as propriedades especificadas em seu arquivo de opções).

Informação adicional acerca do GAUCHO pode ser encontrada na referência [53].

5.2.4 HPDMonitor

Um cliente desenvolvido usando ROOT e DIM (encontrado no pacote HPDMonitor do LHCb) que pode acessar histogramas publicados pelo GAUCHO e criar uma visualização deles em uma interface gráfica. O cliente é capaz de entender o *array* que gera o MonitorSvc e transformá-lo em um histograma do ROOT em tempo real.

5.2.5 Gerente de multi-buffers

O experimento LHCb emprega 2.000 CPUs em seu cluster HLT (Gatilho de Alto Nível - High Level Trigger). Cada nó tem algorítmos de seleção que rodam em uma arquitetura de processamento de dados baseada em buffers de eventos de memória compartilhada.

Esta arquitetura é feita com programas GAUDI *online* chamados *Producers* e *Consumers* que se comunicam por meio de uma área de memória compartilhada gerenciada: o gerente de buffer (BM). O lado esquerdo da figura 5.4 mostra um bloco de construção da arquitetura. Os dados que estes programas podem manipular são *RawEvents*. *Producers* são os programas que declaram *RawEvents* no BM e *Consumers* são os que lêem e fornecem *RawEvents* ao BM. O BM é gerenciado por um programa GAUDI *online* que roda em cada nó, que cria e inicializa a área de memória compartilhada.

A arquitetura tem um outro tipo de bloco de construção chamado Unidade de Transferência de Dados (*Data Transfer Unit*) construída conectando a saída de um *Consumer* à entrada de um *Producer* em um nó remoto usando uma conexão de rede (lado direito da figura 5.4). Esta unidade permite a transferência de eventos e é usada para mover os eventos aceitos do cluster HLT para o sistema de armazenamento e do sistema de armazenamento ao cluster do monitoramento *online* do LHCb.

Consumers e *Producers* são programas GAUDI *online* que podem ser controlados e monitorados com DIM conforme descrito na seção 5.2.2. Se algum *Consumer* executa algorítimos que criam histogramas, eles são publicados usando o MonitorSvc do GAUCHO conforme descrito na seção 5.2.3.

Informação adicional pode ser encontrada nas referênçias [54] e [55]

5.3 O sistema *Online* do MINERνA

Originalment o sistema *online* do MINERνA escria os dados em um disco rígido. Todos os algorítmos do MINERνA operavam em ambiente *offline* onde um processo lia os dados armazenados em disco, processava-os segundo o Modelo de Evento do MINERνA, preenchia histogramas e escrevia os resultados novamente em disco rígido. Em seguida, outros programas

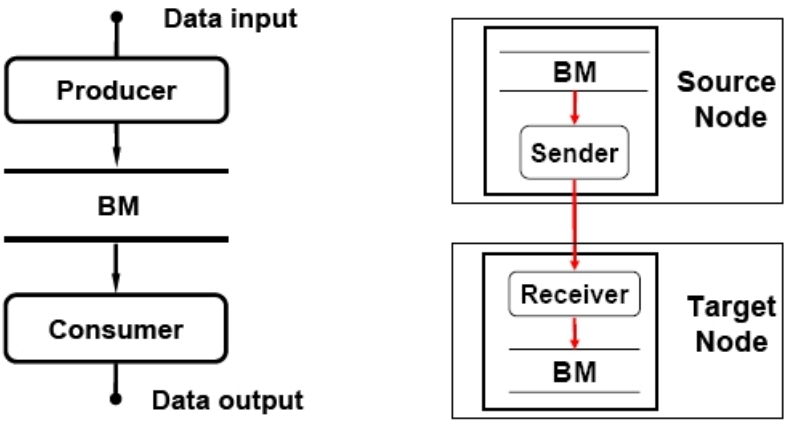

Figura 5.4: Bloco de construção básico para processamento de dados HLT do LHCb e as unidades de transferência de dados capazes de transferir dados entre BM em dois nós diferentes.

(baseados em ROOT) liam os histogramas do disco e os apresentava em tela.

Parte fundamental de nosso trabalho consistiu em integrar todos os processos em um único sistema capaz de realizar todas as etapas em tempo real.

A versão original do sistema online do MINERνA é mostrada esquematicamente na figura 5.5. Consiste de três procesos (programas GAUDI-*online*) cujo objetivo é ler o *hardware* de aquisição de dados e montar uma sequência de dados chamada de *RawEvent* (um objeto que é parte do modelo de evento do MINERνA e LHCb) para depois armazená-los em um disco rígido:

Output Buffer Este processo é um gerenciador de multi-*buffers* e cria um *buffer* chamado *OUT_14d* com tamanho de 8 MB.

PROD_0 Este processo é específico para o experimento MINERνA e foi desenvolvido em c#[1]. Trabalha como um *Producer* do buffer *OUT_14d*. Ele lê dados dos *drivers* do *hardware* e cria a sequência chamada *RawEvent* usando a classe *MinervaProducer* que é uma implementação da classe *Producer* usada no LHCb.

DiskWR Este processo é um *Consumer* do buffer *OUT_14d*. Ele se registra no *buffer* e aguarda a declaração de *RawEvents* pelo *MinervaProducer*. Quando os *RawEvents* estão disponíveis no buffer, o proceso os lê e, empregando o algorítmo GAUDI chamado *DecodeRawEvent*, realiza um processamento transformando os *RawEvents* em *RawDigits* (desembalagem) e escreve tudo em disco rígido.

[1]Uma linguagem de programação simples, moderna e de uso geral desevolvida pela Microsoft para ser utilizada no sistema operativo Windows

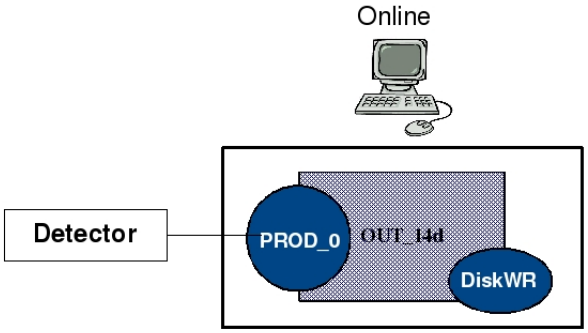

Figura 5.5: Versão original do sistema *online*

Pode-se notar a semelhança do sistema *online* com o bloco de construção básico para processamento de dados usado no HLT do LHCb mostrado na figura 5.4.

Esta verssão do sistema *online* foi utlizada, com êxito, durante as primeiras tomada de dados do Protótipo de Rastreamento e do *ForestTest*. O funcioanamento adequado do sistema no detector completo do MINERνA requer uma minimização do tempo de leitura. Duas condições foram consideradas:

- Empregar as CPU mais rápidas, uma vez que o maior gargalo de velocidade advém de leitura, que é uma atividade muito dependente da velocidade do processador. Podemos fazer algumas mudanças no *hardware/firmware* para diminuir o impacto deste efeito. A maior parte do tempo empregado na leitura do detector ocorre no acesso às memórias do CROC, um processo que é feito basicamente em forma serial por cada CPU. Uma CPU mais rápida, de múltiplos núcleos, é de grande valia neste processo onde as tarefas de I/O e de abertura e fechamento consomem muito tempo e o DAQ.

- Empregar um sistema para leitura do detector e outro sistema em uma LAN para desempacotar os eventos e escrevê-los em disco. Se movermos o desempacotamento de eventos para uma máquina diferente obtemos uma melhoria da ordem de um fator dois. Para isto precisamos quebrar o DAQ em processos capazes de rodarem em dois PC diferentes conectados via LAN.

O desenvovimento da segundo proposta dá lugar a implementação do sistema quase-*online* que é descrito mais adiante e que úm dos objetos desta dissertação .

Independentemente, era preciso iniciar o desenvolvimento do sistema de monitoramento em tempo real do detector. Isto é, também, um dos objetos desta dissertação e descrito adiante.

5.4 Sistema *Online* modificado

Estava planejado que o último processo DiskWR seria o encarregado de rodar os algorítmos geradores de histogramas permitindo o monitoramento em tempo real, como mostrado na figura 5.6.

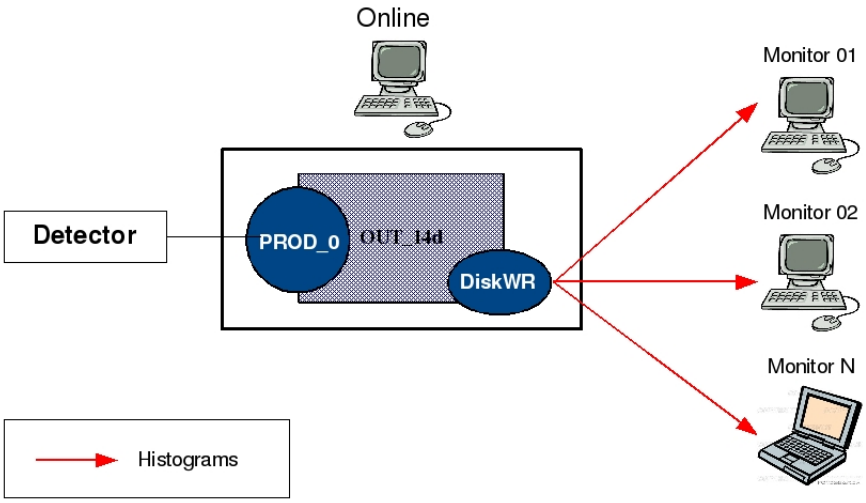

Figura 5.6: O sistema *online* capaz de publicar histogramas para o monitoramento em tempo real do detector

Entretanto este sistema *online* não é viável para emprego com o detector completo pois seu emprego acarreta aumento significativo da tomada de dados. Com o único objetivo de testar as ferramentas do monitoramento em tempo real foi desenvolvido um algorítmo GAUDI simples chamado *Montest* que rodava no ambiente *offline*. O algorítmo é apresentado no anexo E. Parte de meu trabalho consistiu em integrar este algorítmo ao sistema *online* da forma como pode ser vista na figura 5.7.

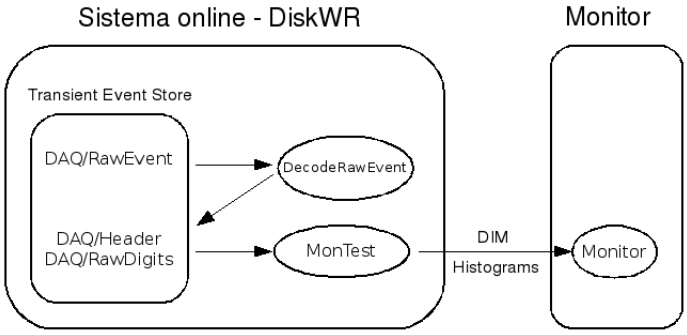

Figura 5.7: O algorítmo *MonTest* foi integrado ao processo DiskWR do sistema *online*

Este algorítmo tem acesso a Aramazenagem Transiente de Eventos (como qualquer outro

algorítmo GAUDI) e lê os *RawDigits* e *DAQHeader* que foram declarados pelo *DecodeRawEvent* (algorítmo de desempacotamento). Com esta informação o *MonTest* é capaz de construir um histograma que mostra a distribuição de carga coletada num evento em contagens de ADC considerando todos os canais das fotomultiplicadoras conectadas. A figura 5.8 mostra uma destas distribuições (visualizada por intermédio do cliente de monitoramento *HPDMonitor* do LHCb) obtida numa tomada de dados do *ForestTest* com 20 caixas fotomultiplicadoras (1280 canais). O algorítmo *MonTest* foi empregado para testar as ferramentas que permitem a publicação e visualização de histogramas.

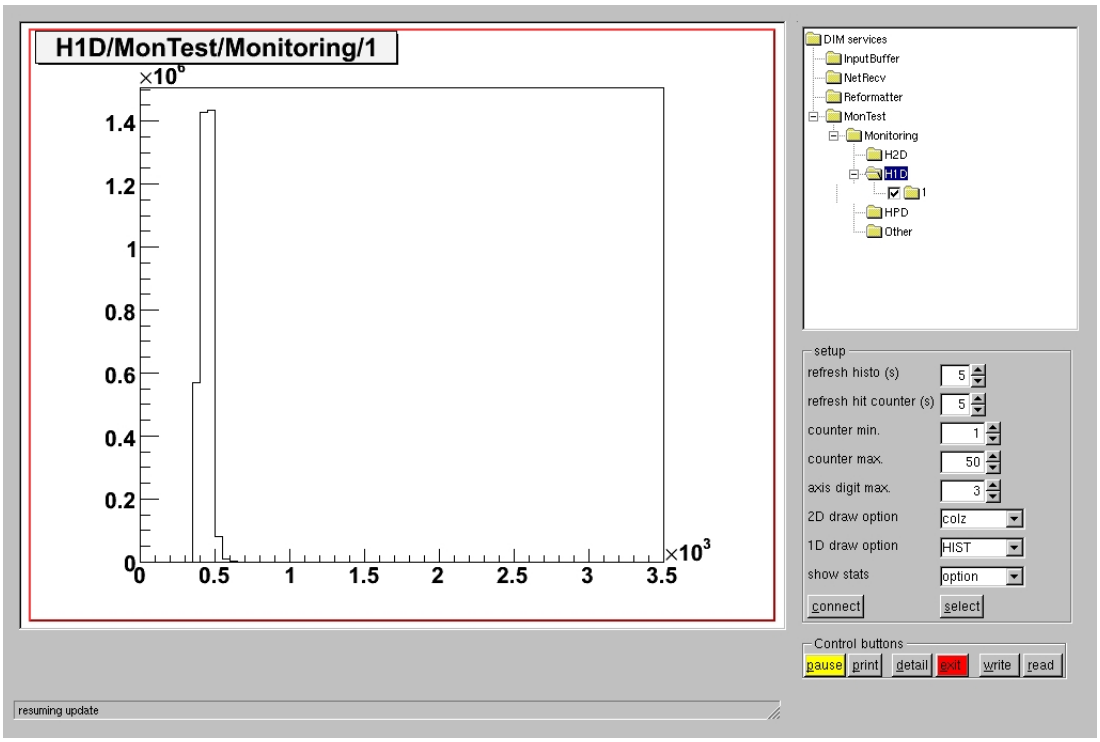

Figura 5.8: Histograma gerado pelo algorítmo *MonTest* no sistema *online* e mostrado pelo *HPDMonitor* do LHCb

O *MonTest* é um algorítmo simples de monitoramento, que requer o desempacotamento dos dados através do algorítmo *DecodeRawEvent* que roda na máquina do sistema *online*. Isto implica em uma redução na velocidade de operação do sistema *online*. Além disto, o *MonTest* realiza um único histograma englobando todos os canais.

Uma vez testadas todas as ferramentas, realizei a implementação do algorítmo *BasicHistos* no sistema *online*. O *BasicHistos* foi desenvolvido e originalmente usado no ambiente *offline* para transformar *RawDigits* e *DAQHeader* em histogramas simples que podem ser visualizados com ferramentas padrão como o ROOT. Ele cria um histograma por cada canal conectado ao detector contendo a distribução de carga coletada em contagens de ADC, pa cada um dios canais. A figura 5.9 mostra alguns destes histogramas gerados pelo *BasicHistos* numa tomada

de dados do detector.

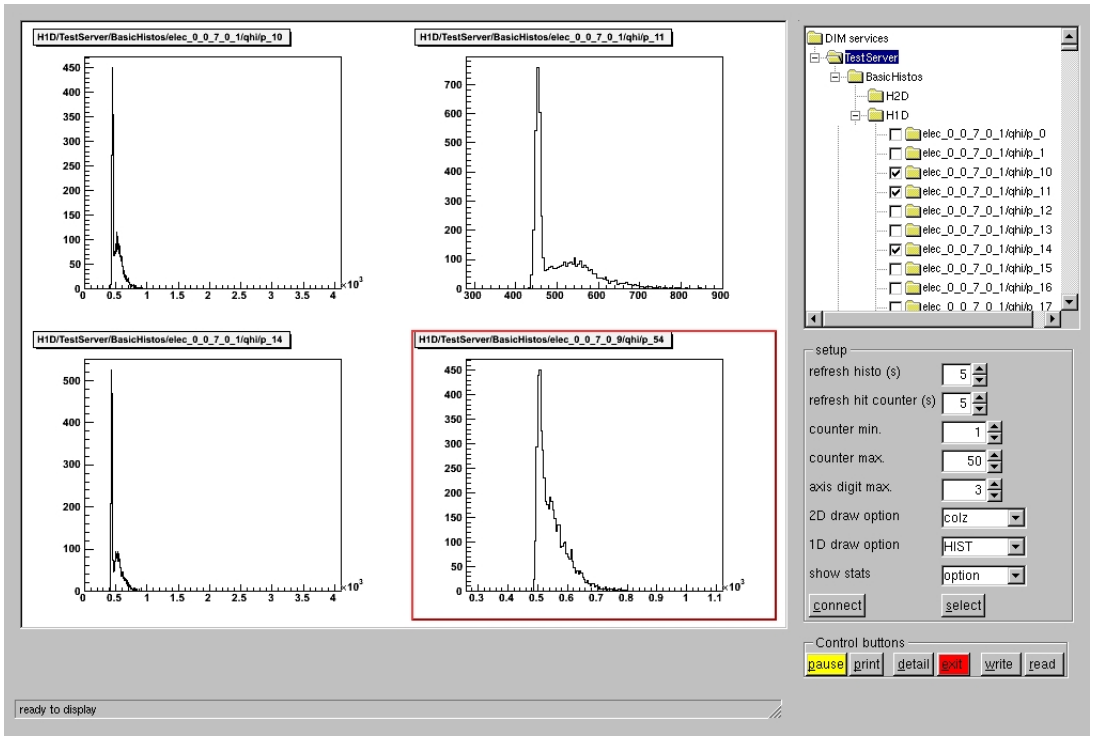

Figura 5.9: Histogramas gerados pelo algorítmo *BasicHistos* no sistema *online* e mostrado pelo *HPDMonitor* do LHCb

5.5 Origem do sistema quase-*online*

Com o objetivo de rodar os algorítmos para o monitoramento em tempo real sem sobrecarregar o sistema *online* modifiquei o processo DiskWR de maneira que ele seja capaz de enviar os objetos *DAQHeader* e *RawDigits* a processos rodando emn outro computador. O sistema (ou conjunto de processos) encarregado do receber e processar os dados foi chamado de quase-*online*.

Para este fim escrevi os algorítmos *PublisherAlg* e *ReceiverAlg* cujos códigos fonte encontram-se no anexo E. O algorítmo *PublisherAlg* publica um serviço DIM que contém toda a informação do evento (*RawDigits* e *DAQHeader*) em um *array* de inteiros de uma maneira similar a como os histogramas são publicados pelo MonitorSvc no pacote GAUCHO (ver seção 5.2.3). O evento é enviado imediatamente após o desempacotamento. O algorítmo *ReceiverAlg* se registra no serviço publicado pelo algorítmo *PublisherAlg* por meio da classe *DimInfoEvent* (também no anexo E) que adaptei da classe *DimInfoHistos* usada pelo LHCb em seu *HPDMonitor* para receber histogramas com DIM. *ReceiverAlg* recebe a informação do evento e a declara na TES local para que seja visível aos algorítmos de monitoramento em

tempo real (como *Montest* e *BasicHistos*) e etágios superiores do processamento de dados (em fase de implementação). A figura 5.10 ilustra o que foi descrito aqui.

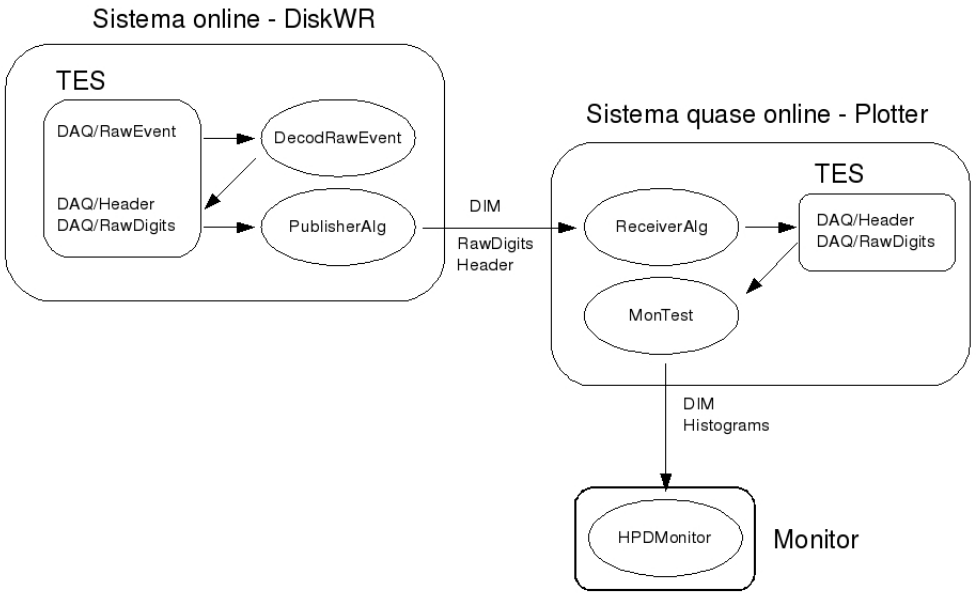

Figura 5.10: O sistema *online* capaz do enviar *RawDigits* e *DAQHeader* através da rede para um processo chamado Plotter rodando no outro computador dedicado ao processamento e monitoramento em tempo real.

Na situação descrita, o sistema *online* não roda os algorítmos para o monitoramento em tempo real, mas precisa fazer o desempacotamento e o envio de dados. Isto não o torna significativamente mais rápido que o descrito na seção 5.4. A solução deste problema foi parte significativa de minha contribuição e esta descrita na seção seguinte.

5.6 Os sistemas *Online* e quase-*online* atuais

Segundo o descrito na secção 5.2.5, *RawEvents* podem ser transferidos mediante uma conexão de rede utilizando o bloco chamado Unidade de Transferência de Dados. Eu implementei este bloco no MINERνA empregando os processos *NetSend* e *NetRecv* como se vê na figura 5.11. Nesta situação não é mais preciso o desempacotamento no sistema *online*, que apenas envia os *RawEvents* para o sistema quase-*online*, onde o desempacotamento é então realizado. Uma pequena descrição dos processos rodando em ambos os sistemas é dada a seguir.

OutputBuffer Este processo não mudou e a descrição dada no 5.3 é ainda válida.

PROD_0 Este processo não mudou e a descrição dada no 5.3 é ainda válida.

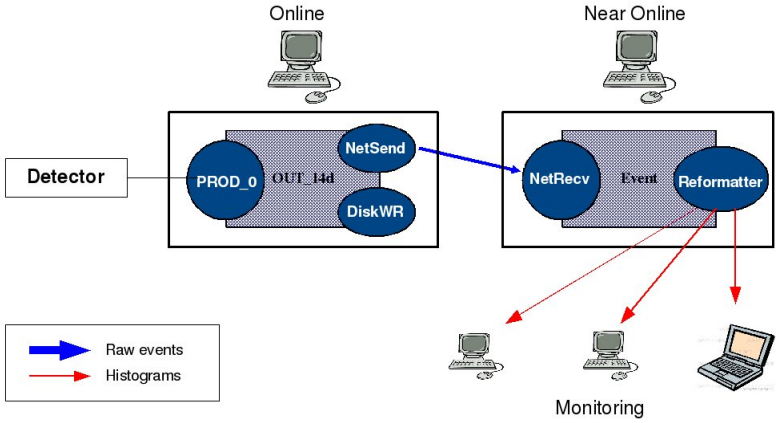

Figura 5.11: Os sistemas *online* e quase-*online*

DiskWR Este processo foi modificado por mim de forma a não mais fazer o desempacotamento de dados. Ele simplesmente escreve os *RawEvents* que são declarados no buffer *OUT_14d* no disco rígido. Eventualmente esta operação será feita no sistema quase-*online*, quando então *DiskWR* ser removido do sistema *online*. O arquivo de opções do GAUDI para este processo, chamado *MinervaDiskWR.opts*, pode ser encontrado no apêndice E.

NetSend Este processo é um novo *consumer* do buffer *OUT_14d* no sistema *online* e lê *RawEvents* do buffer para enviá-los ao proceso *NetRecv* que roda no sistema quase-*online*. Neste processo roda um algorítmo GAUDI chamado *DimDataSender* encarregado de transferir os *RawEvents* usando DIM a um outro computador. O código fonte deste algorítmo pertenece ao LHCb mas as opções de configuração do processo (que é onde os algorítmos são chamados) foram por mim modificadas para as características do MINERνA. Estas opções estão especificadas no arquivo *MinervaNetSend.opts* que pode ser encontrado no apêndice E.

InputBuffer Este processo, igual ao *OutputBuffer* no sistema *online*, é um gerenciador de multi-*buffers* e cria o *buffer* chamado *Event* no sistema quase-*online* com tamanho de 8 MB.

NetRecv Este processo é um *Producer* do *buffer Event* no sistema quase-*online* e recebe os dados enviados pelo processo *NetSend* para depois declará-los ao buffer *Event*. Neste processo não roda nenhum algorítmo GAUDI. Ele recebe os *RawEvents* mediante um serviço GAUDI chamado *DimDataReceiver*. Assim como o processo *NetSend*, o proceso *NetRecv* pertence ao LHCb mas as opções de configuração foram por mim alteradas para atender as necessidades do MINERνA. Estas opções encontram-se no arquivo *MinervaNetRecv.opts* que pode ser visto no apêndice E.

Reformatter Eu implementei este processo baseado no *DiskWR* do MINERνA. Este processo é um *Consumer* do *buffer Event* no sistema quase-*online*. Quando os *RawEvents* são declarados no *buffer*, o processo os lê, faz o processamento de dados (desempacotamento), roda algorítmos de monitoramento e escreve tudo no disco rígido. As opções deste processo encontram-se no arquivo *Reformatter.opts* que pode ser visto no apêndice E.

Para rodar estes processos eu modifiquei o *script runminervadaq.bat* e criei um novo *script* chamado *fsm_status.bat*, ambos descritos nos apêndices E e C. Para iniciar os processos no sistema quase-*online* eu criei o *script daq_recv.sh*, também descrito nos apêndices E e C.

Detalhes de como a Unidade de Transferência de Dados é implementada no MINERνA podem se vistos na figura 5.12. Observe-se como a TES do processo *NetSend* no sistema *online* é transferida ao TES do processo *NetRecv* no sistema quase-*online*.

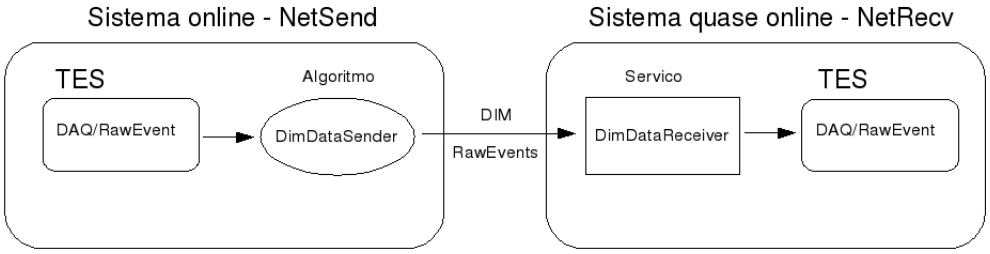

Figura 5.12: O bloco Unidade de Transferência de Dados para o MINERνA

Depois de receber os *RawEvents* na TES do processo *NetRecv* eles são declarados no *buffer Event* para serem lidos pelo processo *Reformatter* e colocados na sua própria TES. Ao final, é o *Reformatter* o processo encarregado de todo o processamento de dados e é, também, o servidor de histogramas para o monitoramento em tempo real. A figura 5.13 ilustra o que é feito pelo processo *Reformatter*. Ele roda o algorítmo desempacotador (*DecodeRawEvent*) e os algorítmos geradores de histogramas *BasicHistos* e *ADCCountsAlg*. O algorítmo *ADCCountsAlg* faz uso de uma classe chamada *ConvertChannelID* para criar histogramas. Ambos os códigos foram desenvolvidos por mim e são apresentados no apêndice E.

É necessário adaptar ao sistema quase-*online* todos os algorítmos usados no ambiente *offline* para o processamento de dados e desenvolver algorítmos geradores de histogramas em cada estágio. Isto está em desenvolvimento tendo eu, até o momento, desenvolvido um novo algorítmo gerador de histogramas, chamado *HitCountsAlg* que, junto com a classe *ConvertChannelID*, pode gerar histogramas a partir de *SupDigits*. O código do algorítmo pode ser encontrado no apêndice E. Os objetos chamados *SupDigits* são gerados pelo algorítmo *PedSuppression* a partir de *RawDigits*.

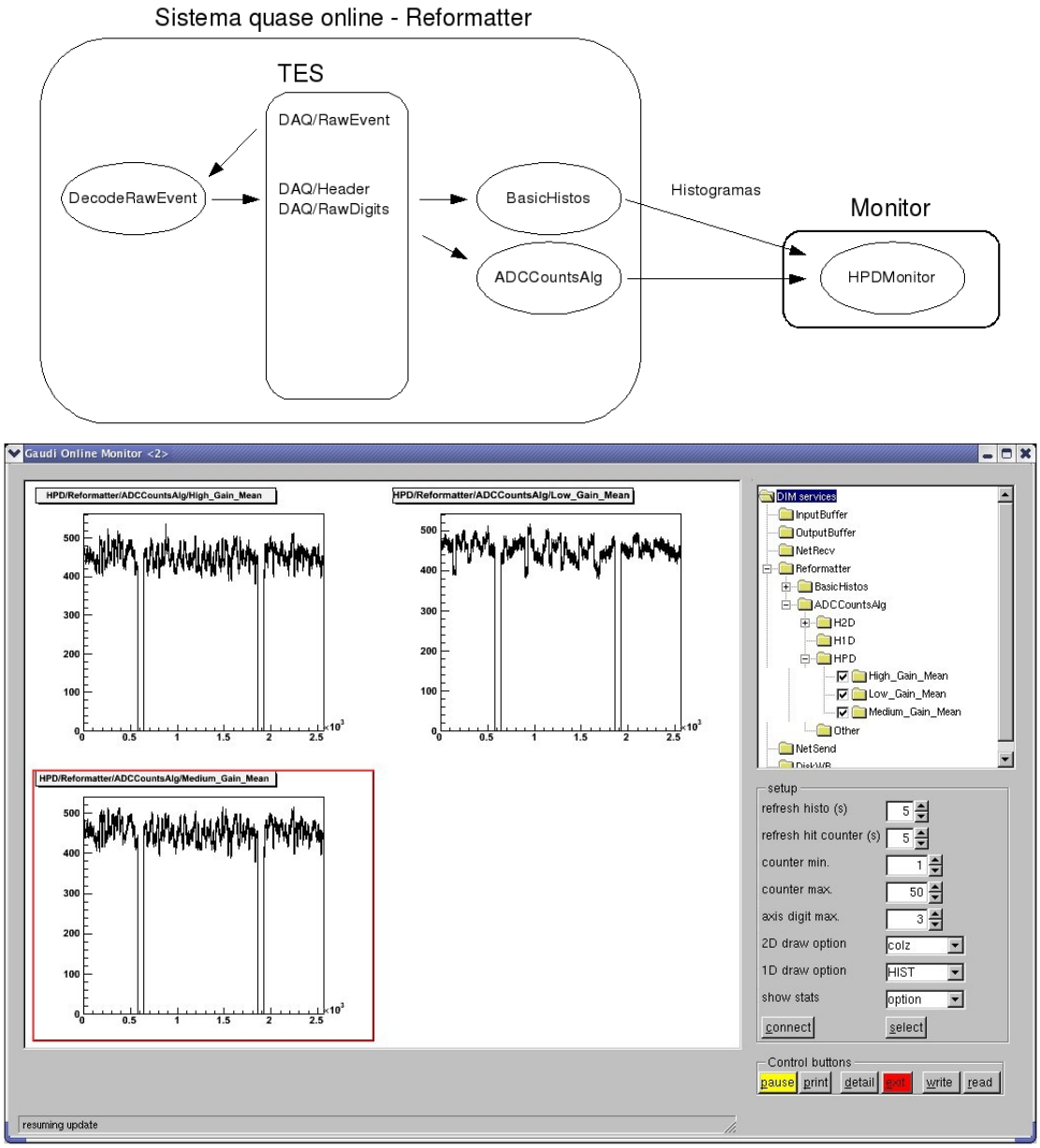

Figura 5.13: O processo *Reformatter* do sistema quase-*online*. O *HPDMonitor* do LHCb mostra alguns dos histogramas resultantes

O algorítmo *PedSuppression* suprime os pedestais dos *RawDigits* usando um arquivo gerado por um outro algorítmo chamado *PedFinder* que só deve rodar com eventos tipo *Pedestal*[2]. Quando ele armazena 1.000 eventos deste tipo o arquivo que o *PedSuppression* precisa é criado, contendo uma tabela com a posição e forma da distribução de pedestal (seu valor médio e desvio padrão) para cada canal das caixas fotomultiplicadoras. Com esta informação o *PedSuppression* é capaz de separar a distribução de pedestal e criar os *SupDigits* com os canais que receberam

[2]Eventos deste tipo são os obtidos sem que haja qualquer sinal de luz nas PMT. Este tipo de evento, assim como outros, são vistos no capítulo 6

uma quantidade de luz significativa. Tudo isto é feito *offline* mas a figura 5.14 mostra como pede ser feito quase-*online*.

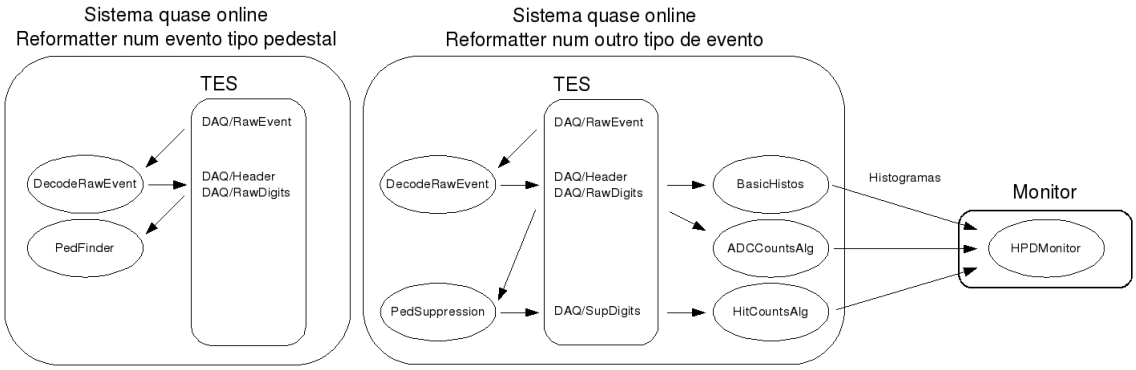

Figura 5.14: Exemplo de como o processo *Reformatter* poderia rodar o algorítmo *PedFinder* nos eventos tipo *Pedestal* e o algorítmo *PedSuppression* nos demais tipos de eventos.

Com os *SupDigits* na TES do Reformatter, o algorítmo *HitCountsAlg* pode ser usado como se ilustra na figura 5.14. Um dos histogramas criados por este algorítmo é mostrado na figura 5.15. Ele mostra o número de vêzes, numa tomada de dados, que cada canal das PMT apresentam sinal acima do pedestal.

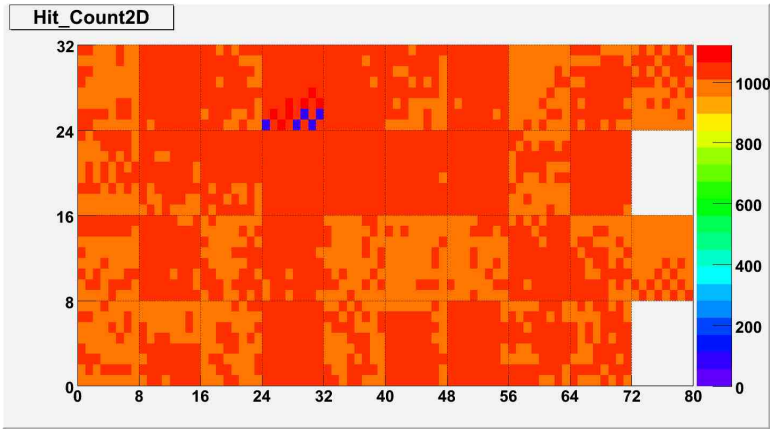

Figura 5.15: O histograma criado pelo algorítmo *HitCountsAlg*. Este gráfico foi criado *offline* pois o algorítmo ainda não está integrado ao sistema quase-*online*.

Todos os códigos desenvolvidos foram incluídos no projeto NEARONLINE e são , agora, parte integrante do *software* do MINERνA. Em complemento ao trabalho aqui apresentado escrevi dois documentos internos para a colaboração MINERνA:

Minerva's Data Acquisition Software: este documento descreve o sistema *online* e o sistema quase-*online*. Este documento pode ser visto, em sua versão portuguesa, no apêndice C.

Minerva's Near Online Manual: este documento é um manual para o usuário e administrador do sistema. Este documento pode ser visto, em sua versão portuguesa, no apêndice D ou em sua versao original na referência [56].

O capítulo 6 apresenta os testes realizados com um protótipo do detector do MINERνA. Estes teste foram usados para testar todas as partes do detector em desenvolvimento, tanto *hardware* como *software*. Participei ativamente desta etapa onde os códigos por mim desenvolvidos foram testados em situação de uso real com tomada de dados com raios cósmicos e outras descritas no capítulo.

Capítulo 6

Protótipo: testes do *software* e do *hardware*

6.1 Introdução

A construção do detector do MINERνA é antecedida pela construção de um detector protótipo chamado de Protótipo de Rastreamento (*Tracking Prototype*) ou simplesmente *TP*. Ele é empregado para testes de todo o sistema (*hardware*, *software*, aquisição de dados) e vem a ser, na verdade, uma parte do detector MINERνA equivalente ao 20% do mesmo. A figura 6.1 mostra a composição do *TP* onde se podem identificar as seguintes partes:

- 10 módulos totalmente ativos para rastreamento
  - 2 planos cintiladores de 17 mm cada um.
- 10 módulos do calorímetro eletromagnético (ECAL)
  - 2 chapas de chumbo de 2 mm cada uma.
  - 2 planos cintiladores de 17 mm cada um.
- 4 módulos do calorímetro hadrônico (HCAL)
  - 1 chapa de aço de 25 mm.
  - 1 plano cintiladore de 17 mm.

O *TP* completo tem um comprimento mínimo de aproximadamente 0,9 m e foi, inicialmente, construido no laboratório WideBand do Fermilab. A medida que ele vai sendo montado ele vai sendo testado e comissionado usando um sistema de injeção de luz e raios cósmicos (múons). Quando estiver completo e testado no WideBand ele será desmontado e instalado novamente na

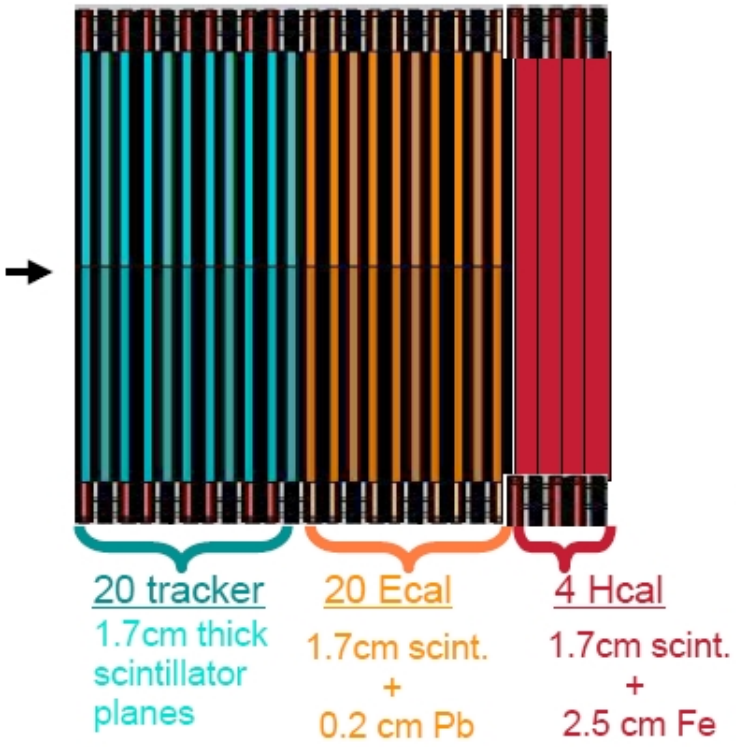

Figura 6.1: O *Tracking Prototype* consiste de diferentes grupos de planos que, ao final, pertencerão ao detector do MINERνA

caverna do MINOS a, então , usado para testes com o feixe NuMI. O *TP* evoluirá, então, para o detector do MINERνA com o acréscimo dos módulos restantes (sempre seguido de testes). Os principais objetivos da construcção do *TP* podem ser resumidos como:

- Testar o projeto do detector.
- Desenvolvimento dos procedimentos operacionais de construcção , instalação e montagem.
- Desenvolvimento dos procedimentos e técnicas de calibração .
- Testar a integração dos diferentes componentes do *hardware* e *software*.
- Testar as diferentes partes do *software* (aquisição de dados, processamento de dados *offline* e *online* , sistemas *online* e quase-*online* e o monitoramento em tempo real).
- Observação e reconstrução do eventos de raios cósmicos.
- Demonstração da capacidade de rastreamento com raios cósmicos e de adquirir e analisar dados com feixe de neutrinos.

Este capíulo trás uma descrição da instalação e tomada de dados com os primeiros módulos do *TP*, tarefas nas quais eu participei ativa e constantemente. São , também, apresentados alguns resultados preliminares conseguidos demostrando a funcionalidade do *TP* e cumprindo alguns dos objetivos mencionados anteriormente, principalmente no que concerne ao trabalho desenvolvido nesta dissertação .

6.2 Montagem dos módulos

A montagen do *TP* é feita um módulo por vez, que são cabeados e instrumentados em grupos de quatro. São necessárias 19 caixas fotomultiplicadoras (PMTs) para os módulos ativos e calorímetro eletromagnético (ECAL) e 11 PMTs par os módulos do calorímetro hadrônico (HCAL). Todas as PMTs são alimentados por uma fonte de alimentação dedicada e conectados ao sistema de injeção de luz. A figura 6.2 mostra o *TP* com os primeiros 4 módulos e seus 19 PMTs instalados e instrumentados no WideBand.

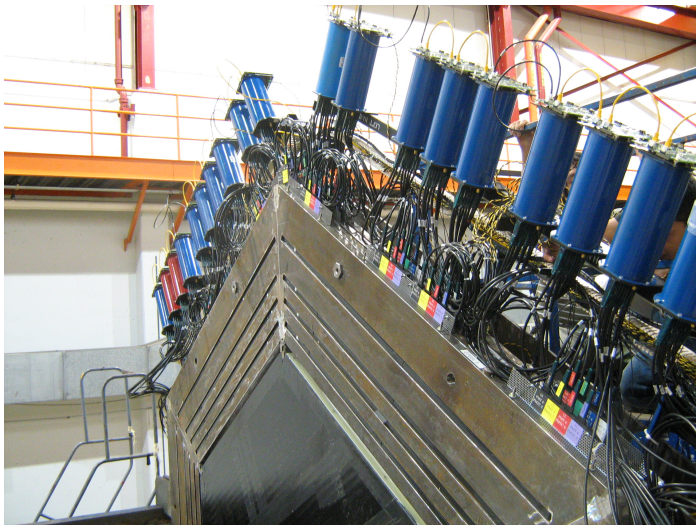

Figura 6.2: O estado do *TP* em setembro de 2008 com os primeiros 4 módulos instalados e instrumentados

A intrumentação do primeiro grupo de módulos foi a mais difícil e demorada já que serviu para o desenvolvimento dos procedimentos de instalação e teste do *TP*. Detalhes da eletrônica envolvida são apresentados na seção 6.3. Uma vez instrumentados, eles são testados usando o sistema de injeção de luz. Com estes dados são calculados o nível de luz detectado e o ganho de cada canal dos PMTs. Adicionalmente, planos cintiladores, que podem ser facilmente colocados e retirados em qualquer momento, são colocados à jusante e à montante do *TP* e utilizados como gatilho para raios cósmicos que atravessam completamente o detector. Estes planos são vistos junto ao *TP* na figura 6.3. Após os testes e tomada de dados, a instalação dos

módulos é reiniciada e o ciclo repetido até que o *TP* esteja completo. A figura 6.4 mostra o *TP* com 20 módulos instalados e instrumentados faltando apenas os 4 módulos correspondentes ao calorímetro hadrônico (HCAL).

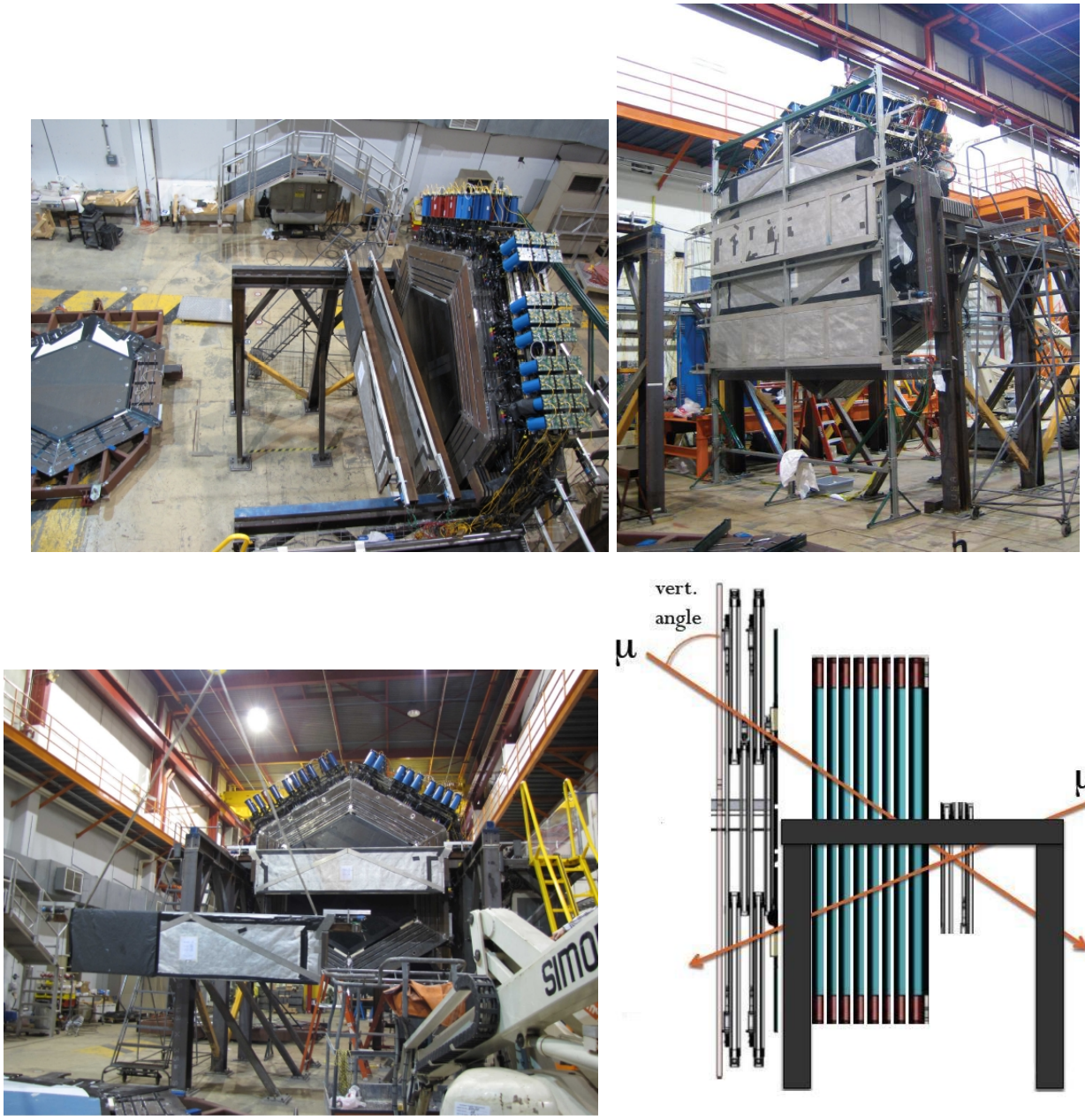

Figura 6.3: Visão do *TP* junto aos planos cintiladores usados para gatilho para sinais de raios cósmicos que atravessam o detector

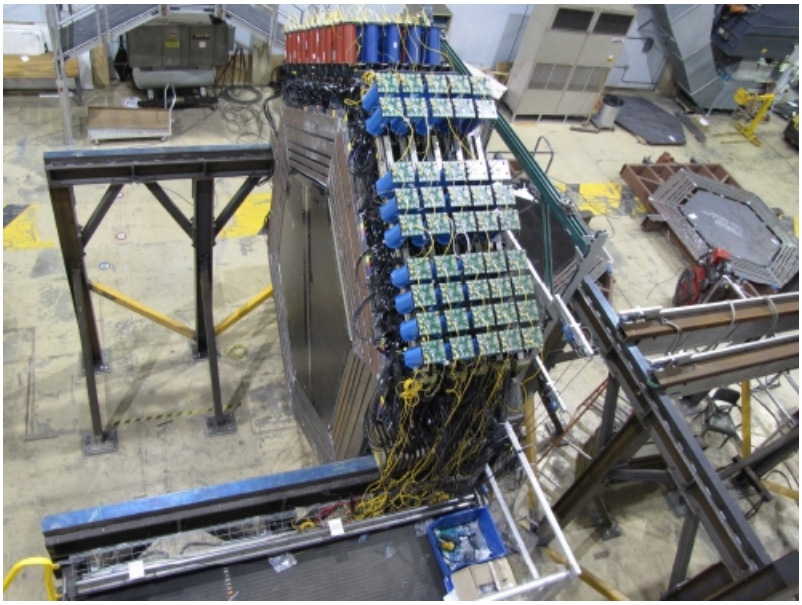

Figura 6.4: O estado do *Tracking Prototype* no final de fevereiro do 2009 com 20 módulos instalados e instrumentados

6.3 Electrônica usada no *TP*

A eletrônica de aquisição de dados usada no *TP* é a mesma que será utilizada no detector MINERνA. Ela é introduzida no capítulo 3 é descrita nesta secção com mais detalhes.

6.3.1 Caixa fotomultiplicadora (PMT)

Dentro dela encontra-se a fotomultiplicadora multi-anodo de 64 canais. Ela contém, também, uma fonte de alta voltagem *Cockcroft-Walton* de 22 estágios que alimenta a fotomultiplicadora.

Cada grupo de quatro módulos emprega 19 PMTs (1216 (19x64) canais). Cada módulo ativo e o eletromanetico contêm 2 planos com 152 tiras cintiladoras das quais 128 pertenecem ao *Inner Detector*(ID) e 24 ao *Outer Detector*(OD) . O detector MINERνA emprega cerca de 500 PMTs enquanto o *TP* usa apenas 105 PMTs.

6.3.2 Sistema de Injeção de luz (LI)

Cada PMT é iluminado com 2 fibras ópticas para os testes com injeção de luz, como mostrado no lado esquerdo da figura 6.5. Testes simples deste tipo, chamados de *ForestTest*, são feitos para todas as PMTs, em grupos de 19, antes de que elas sejão instaladas no detector. O objetivo destes testes é simplesmente verificar o funcioamento dos PMTs. Pode-se ver, no lado direito da figura 6.5, os 8 conectores que contêm 8 fibras ópticas cada um conectados aos PMTs. Estos conectores carregam a sinal de luz provenientes das tiras cintiladoras.

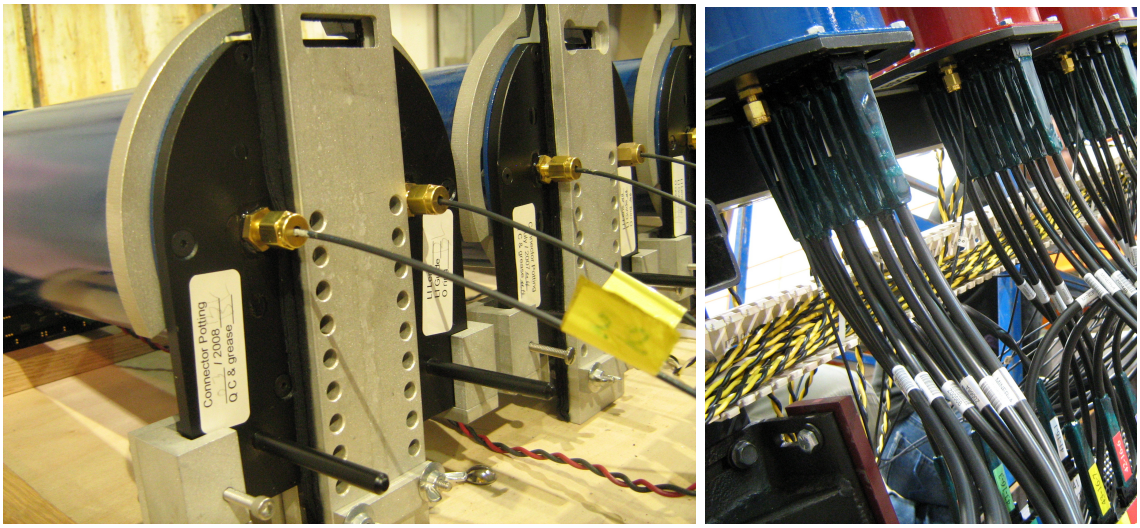

Figura 6.5: O sistema de injeção de luz nos PMTs. O lado direito mostra como as fibras ópticas provenientes do sistema de injeção de luz e das tiras cintiladoras são colocadas no *TP*

19 PMTs são conectadas ao sistema de injeção de luz por meio de fibras ópticas, agrupadas em grupos de 38 fibras. Cada um destes grupos ilumina 19 PMTs de maneira que cada PMT é iluminada por 2 fibras ópticas. O sistema de injeção de luz é mostrado na figura 6.6.

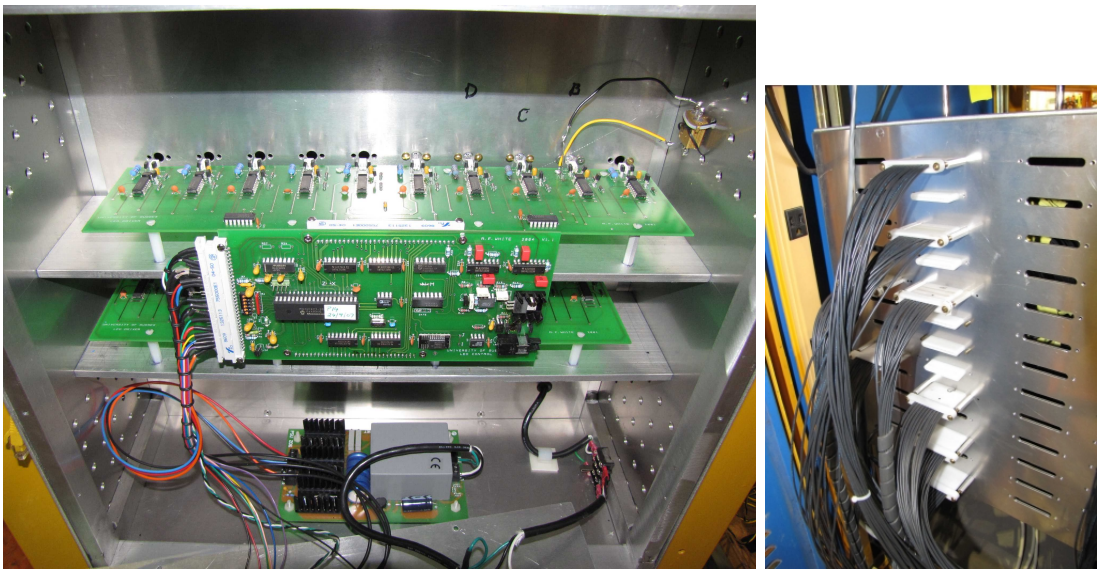

Figura 6.6: O lado esquerdo mostra a eletrônica no interior do sistema de injeção de luz. O lado direito mostra grupos de 38 fibras ópticas que são conectados aos PMTs

6.3.3 Front End Boards (FEB)

É a placa mais importante do *hardware* do experimento MINERνA e existe uma conectada a cada PMT usada no MINERνA ou no *TP*. O projeto original é da colaboração DØ no Fermilab,

tendo sido adaptado para o experimento MINERνA. A placa é usada para amplificação, digitização da amplitude (ADC) e tempo (TDC) do sinal e para o controle da alta voltagem das fotomultiplicadoras. Ela pode ler cada canal com 3 ganhos diferentes e com 10 bits de resolução. Para ler os 64 canais de PMTs, os FEB possuem 192 (64x3) canais ADC e 64 canais TDC.

Grupos de até 12 FEBs, chamados de *cadeias*, podem ser conectadas em serie mediante o protocolo chamado LVDS. Na prática, a metade das PMTs são conectadas em grupos de 9 FEBs e a outra metade em grupos de 10 FEBs de maneira que cada 2 *cadeias* contêm 19 FEBs (PMTs). Assim, pode-se instrumentar grupos de 4 módulos com apenas 2 *cadeias*. Isto pode ser visto nas figuras 6.2 e 6.4. A figura 6.7 mostra uma *cadeia* e pode-se observar como os FEBs são conectados. Para o detector MINERνA são necessárioa 500 FEBs aproximadamente os quais podem ser agrupados em 52 *cadeias*. O *TP* usa 105 FEBs agrupados em 11 *cadeias*.

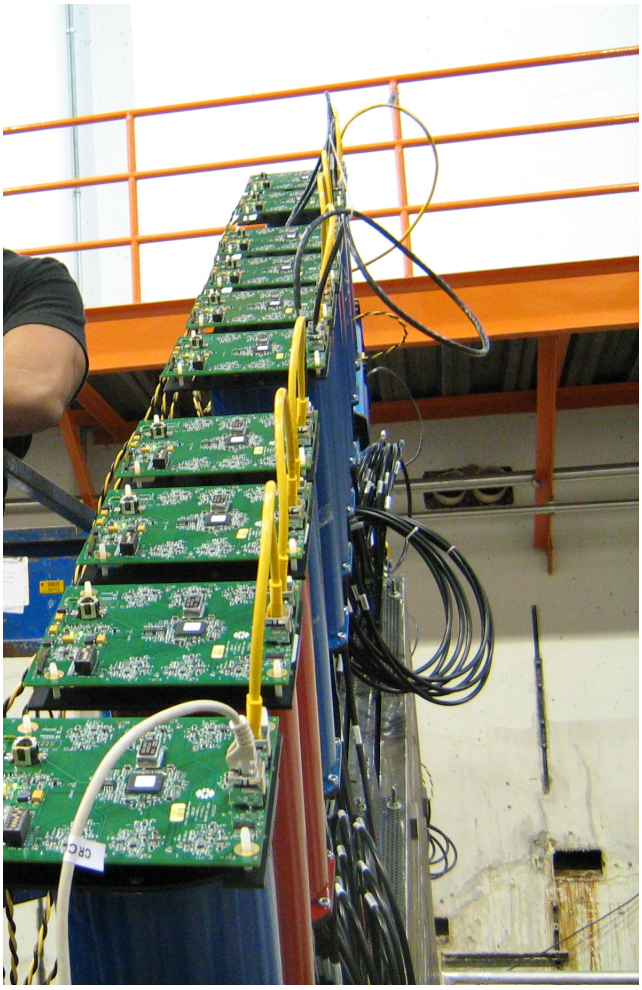

Figura 6.7: Uma *cadeia* de 10 FEB instalada no *TP*.

6.3.4 Front End Support Board (FESB)

São conversores DC-DC que permitem a distribução de baixa voltagem (4VDC) a todos os FEBs em uma *cadeia* a partir dos 48 VDC fornecidos pela fonte de alimentação principal.

6.3.5 Chain Readout Controler (CROC)

É um módulo VME projetado e fabricado pela colaboração MINERνA. A ele são conectadas as *cadeias* de FEBs. Eles são *buffers* de memória que armazenam comandos para serem enviados às *cadeias* de PMTs e dados para serem lidos pelo PC. Cada CROC pode ler 4 *cadeias* em paralelo. São necessários 13 CROCs para ler todo o detector mas apenas 3 CROCs para o *TP*. Mais informação acerca dos módulos CROC pode ser encontrada na referência [57].

6.3.6 CROC Interface Module (CRIM)

O módulo CRIM é uma interface para os módulos CROC que contêm os conectores para os *triggers* usados para abrir o *gate* de leitura e ativar o pulsador do LED para a injeção de luz entre outras coisas. Mais informação acerca dos módulos CRIM pode ser encontrada na referência [58]

6.3.7 Minerva Timing Module (MTM)

É um módulo especial encarregado de distribuir o signal de tempo do NuMI aos CROCs. Ele envia um sinal de *gate* que é o gatilho que dá início à leitura de todos os PMTs.

6.3.8 Bastidor VME

O bastidor VME usado no experimento MINERνA pode abrigar até 15 módulos. Os 13 módulos CROC que o detector MINERνA precisa serão distribuídos por 2 bastidores VME para simplificar o cabeamento e melhorar a velocidade da leitura de dados. Cada bastidor VME alojará, no máximo, 9 módulos (7 CROC, 1 CRIM, 1 MTM). O *TP* só emprega 1 bastidor VME. A figura 6.8 mostra o bastidor VME empregado no *TP* junto a diversos outros componentes.

6.4 Tomada de Dados com o *TP*

O conjunto de sinais, em contagem de ADC, coletados a partir da leitura de todos os canais presentes no detector é chamado de *evento*. A eletrônica do *TP* tem diferentes configurações que permitem adquirir diferentes tipos de *eventos* entre os quais citamos:

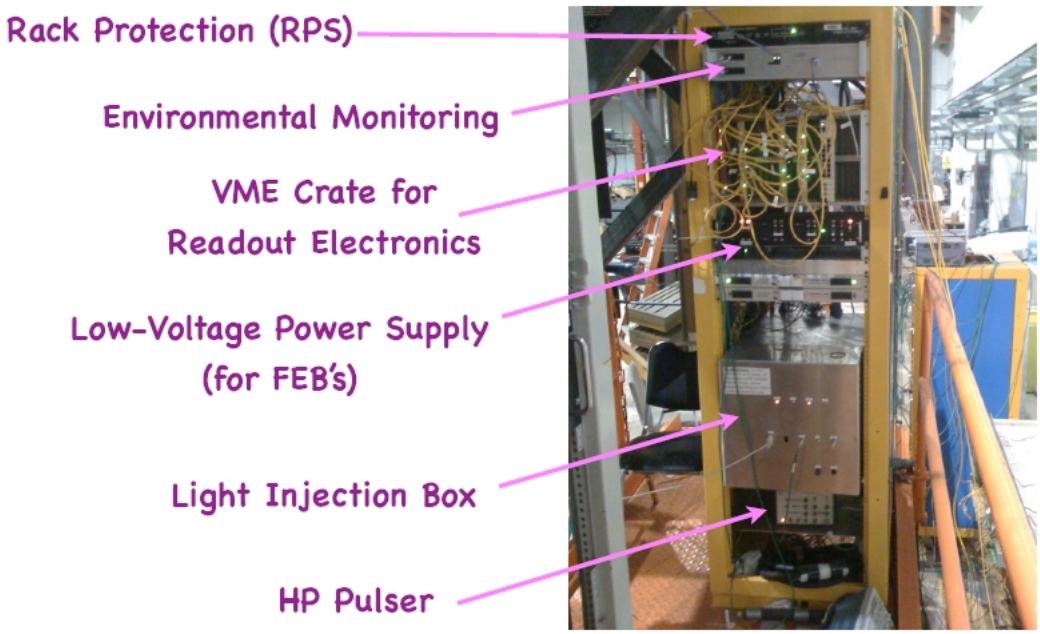

Figura 6.8: O bastidor VME empregado no *TP*

Pedestal Neste modo o *gate* de leitura do detector é aberto durante 10 μs. Imediatemente depois que toda a informação é lida, o *gate* é aberto novamente repetindo-se o ciclo. Os sinais obtidos neste tipo de evento são gerados na ausência de luz nas PMT. Algumas possíveis fontes destes sinais, chamados de ruído, são : emissão de termo-elétrons desde o fotocatodo; corrente de fuga no circuito de anodo da PMT; radioatividade.

Ligth Injection Este modo é muito parecido com o anterior com a única diferença que todos os canais das caixas fotomultiplicadoras são iluminados mediante o sistema de injeção de luz. Este modo permite fazer-se a calibração de ganhos dos canais dos PMTs.

Cosmic Neste modo o detector é colocado entre planos de cintiladores adicionais que são usados para gatilho de raios cósmicos (múons cósmicos). O *gate* de leitura do detector é aberto com uma frequência de 3 kHz e com uma duracão de 10 μs. Apenas os eventos que disparam o gatilho e encontram o *gate* aberto são lidos.

NuMI É o modo a ser utilizado pelo MINERνA e o *TP* durante tomada de dados com o feixe de neutrinos NuMI. Neste modo o *gate* de leitura do detector é aberto a cada 2 segundos (0,5 Hz) aproximadamente com una duracao de 10 μs. O sinal de *gate* é fornecido pelo módulo MTM que se encontra no bastidor VME.

Os eventos são coletados por um computador dedicado a esta tarefa que roda o *software* de aquisição do dados do experimento MINERνA. O conjunto de vários *eventos* é chamado de *run*

que é guardado em disco rígido. É possível ter *runs* de Pedestal, *runs* de Light Injection, *runs* Cosmics ou *runs* NuMI, dependendo dos eventos contidos. Todos os *runs* são aproveitados para testar a eletrônica, o *software* do sistema de adquisição de dados (DAQ), o processamento de dados *offline* e *online* e o sistema de monitoramento em tempo real.

Os arquivos gerados pelo *software* de adquisição de dados têm um formato especial compatível com o modelo de *software* adotado pela colaboração MINERνA para processamento de dados. Estes dados são processados pelo sistema quase-*online* atá a etapa em que os *RawDigits* são criados. Histogramas são criados, a partir dos *RawDigits*, pelo algorítmo *BasicHistos*. Estes histogramas podem ser minitorados como mostrado na figura 5.9 do capítulo 5. Ao final do processamento o sistema quase-*online* cria dois arquivos:

- O arquivo *_RawDigits.root[1] contendo os objetos de dados *DAQHeader* e *RawDigits*.
- O arquivo *_RawHistos.root contendo todos os histogramas gerados pelo *BasicHistos*.

Deste ponto em diante os dados são processados *offline* segundo o tipo de evento que contenham. Eu participei do processamento destes dados, o que permitiu-me familiarizar-me com as diferentes versões do *software* do MINERνA . O processamento online de dados, que permite o monitoramento, está sendo implementado e suas bases foram desemvolvidas em meu trabalho.

Detalhes sobre o *software* do MINERνA e sobre o processamento *offline* são apresentados no capítulo 4 e o caso especial de processamento *online* , e monitoramento da qualidade deles em tempo real, que constitui a maior parte de meu trabalho, é apresentado no capítulo 5. As seções seguintes apresentam alguns resultados do processamento de dados feito segundo o diagrama da figura 4.6 do capítulo 4.

6.4.1 Runs de Pedestal

Neste tipo de tomada de dados (*run*), normalmente são tomados não menos de 1.000 eventos de pedestal que servem para identificar e caracterizar a distribução do sinal na ausência de luz. A figura 6.9 mostra a distribuição de pedestal para um elemento de PMT obtido do arquivo *RawHistos*.

Estes tipos de *runs* prestam-se, também, à identificação de infiltração de luz (*light leak* nas coneções ópticas do TP durante seu comissionamento. Participei desta atividade em conjunto com outros mebros da colaboração empregando um *script* em ROOT que trabalha *offline* nos arquivos *RawHistos* identificando canais nas PMTs com sinais de pedestal atípicos (valores altos

[1]O símbolo * no nome do arquivo representa parâmetros úteis para identificar o conteúdo como o número do *run*, data de criação, tipo de *run*, versão do *software* utilizada entre outros

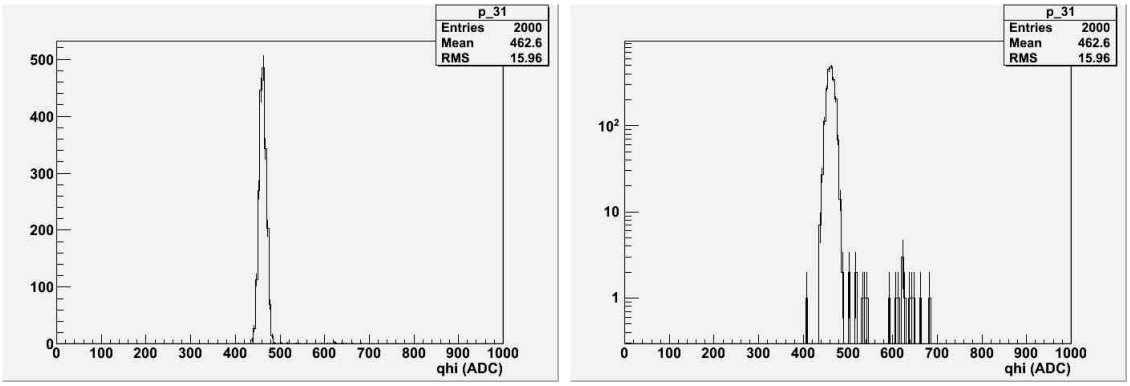

Figura 6.9: Uma distribução de pedestal comum obtida a partir de um só canal das PMTs. A amplitude do sinal (carga coletada) é apresentada em contagens de ADC. O lado direito apresenta a mesma distribuição em escala logarítmica.

de RMS). Uma vez identificado o canal, a origem da infiltração é procurada e anulada. Novos *runs* de pedestal são realizados e o teste repetido até que todos os *ligth leaks* sejam anulados. Uma exemplo de como os *ligth leaks* podem ser identificados qualitativamente é mostrado na figura 6.10.

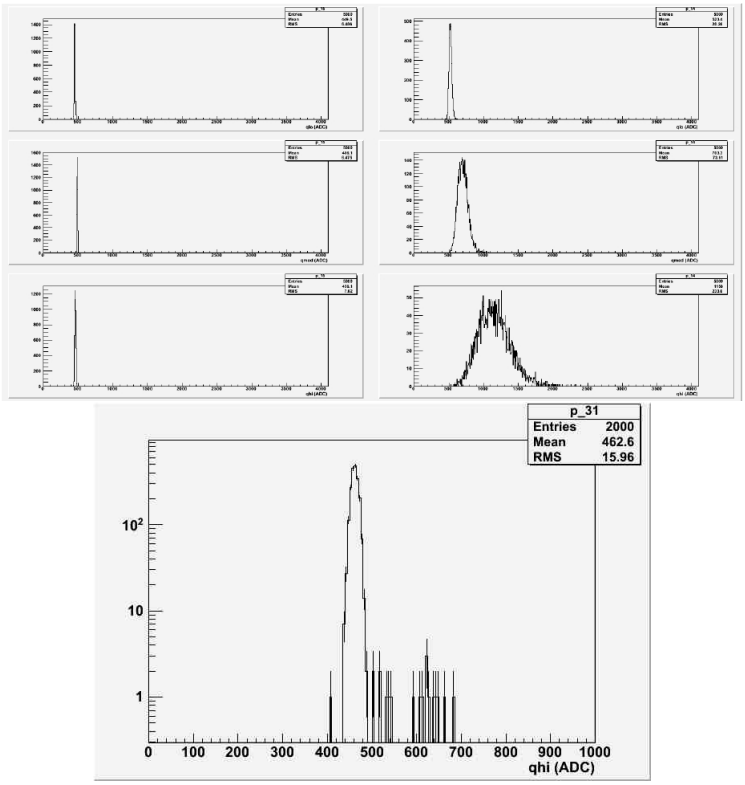

Figura 6.10: Na esquerda da figura pode-se observar um canal do PMT típico. À direita é mostrado um canal com *ligth leak*.

6.4.2 Runs de Ligth Injection

Os *Runs de Ligth Injection* são feitos para calcular os ganhos e a intensidade de luz em cada canal das PMTs permitindo testá-las e caracterizá-las. Nestes testes, luz proveniente de um LED é injetada em cada canal das PMTs. Detalhes deste modelo podem ser encontrados na referência [59].

Cada grupo de 19 PMTs que é instrumentado e montado no *TP* passa por este teste que é feito com diferentes valores de voltagem nos PMTs e com diferentes valores de intensidade de luz. Os diferentes valores de voltagens servem para encontrar o valor ótimo (perto de 800 V) para cada PMT. Assim, cada PMT terá um valor de voltagem único e particular. Os *Runs Ligth Injection* são processados quase-*online* da mesma forma que os *Runs de Pedestal*. Após obter os dois arquivos (*RawHistos* e *RawDigits*), o processamento é feito *offline*. As diferentes distribuições obtidas nestes *runs* são mostradas na figura 6.11 obtidos do arquivo *RawHistos*.

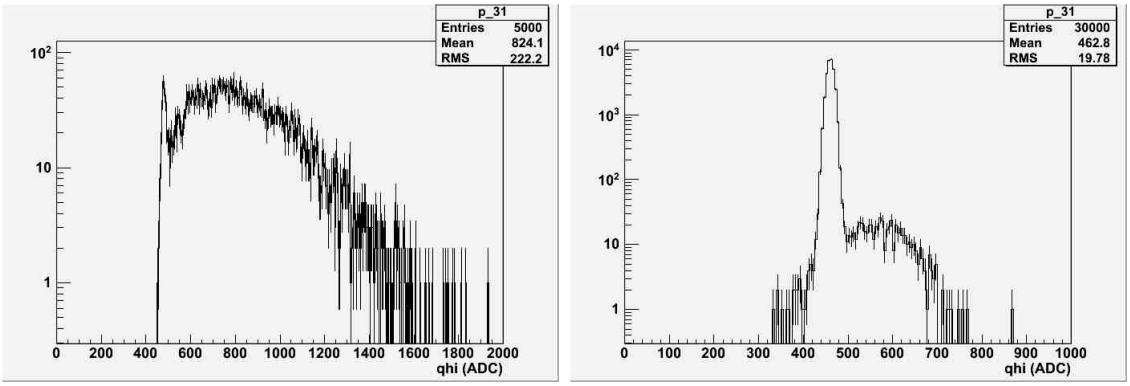

Figura 6.11: Distribuções obtidas com *run de Ligth Injection*

Para cada voltagem nos PMTs (700V ,800 V, 900 V, 750 V, 850 V) são tomados dados segundo a seguinte sequência:

- 2.000 eventos ZeroPE (Pedestal)
- 5.000 eventos MaxPE
- 3.0000 eventos OnePE
- 1.000 eventos ZeroPE (Pedestal)

ZeroPE refere-se aos eventos de pedestal descritos anteriormente e que são usados basicamente com o objetivo de caracterizar a distribução de pedestal.

MaxPE refere-se a maior intensidade de luz possível injetada pelo sistema de injeção de luz e é utilizado para calcular os ganhos de cada canal e, como resultado secundário, é obtido

também a intensidade de luz incidente no canal do PMT em PEs. Este trabalho emprega o arquivo *RawHistos* obtido em um *run de light injection* criado pelo sistema quase-*online* .

Estes resultados, apresentados na figura 6.12, são guardados numa base de dados usando a technologia MYSQL.

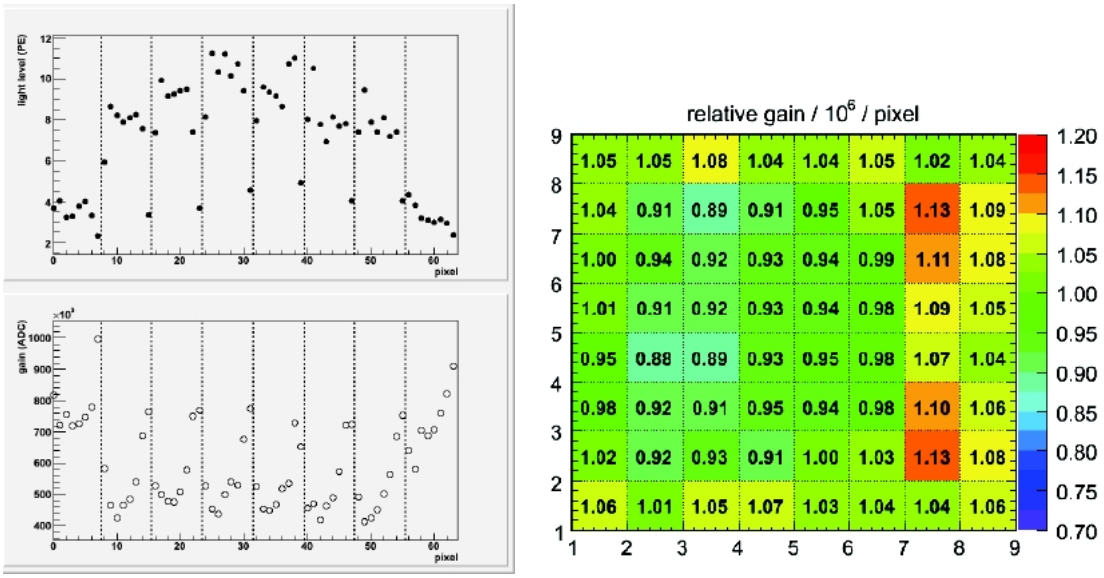

Figura 6.12: Resultados obtidos a partir dos *runs de Ligth Injection* usando a distribução de pedestal e de MaxPE.

OnePE é a intensidade de luz aproximadamente igual a 1 PE e é usado para um estudo mais detalhado da resposta dos PMTs caracterizando a distribução de pedestal e o sinal de luz. O resultado deste trabalho é mostrado na figura 6.13 e os fundamentos deste procedimento, assim como o modelo usado para descrever as fotomultiplicadoras, podem ser encontrados na referência [59]. Este trabalho emprega o arquivo *RawHistos* criado pelo sistema quase-*online* .

.

6.4.3 Runs Cosmics

Nos *runs Cosmics* cerca de 2.000 eventos do tipo *Cosmic* são acumulados durante uma hora. Após o processamento quase-*online* , o arquivo contendo *RawDigits* é processado *offline* segundo o diagrama da figura 4.6.

Para este processamento empregram-se a distribuição de pedestais (na criação de *SupDigits*) e as constantes de calibração de ganho (na criação dos *IDDIgits* e *ODDigits*), obtidas a partir dos *runs* de pedestal e *runs de ligth injection* respetivamente. Após isto, pode-se obter as trajetórias de raios cósmicos usando o algorítmo *zTracker*. O resultado do procesamento é guardado em dois arquivos: um arquivo em formato ROOT, de fácil leitura e usado para analise dos dados, e

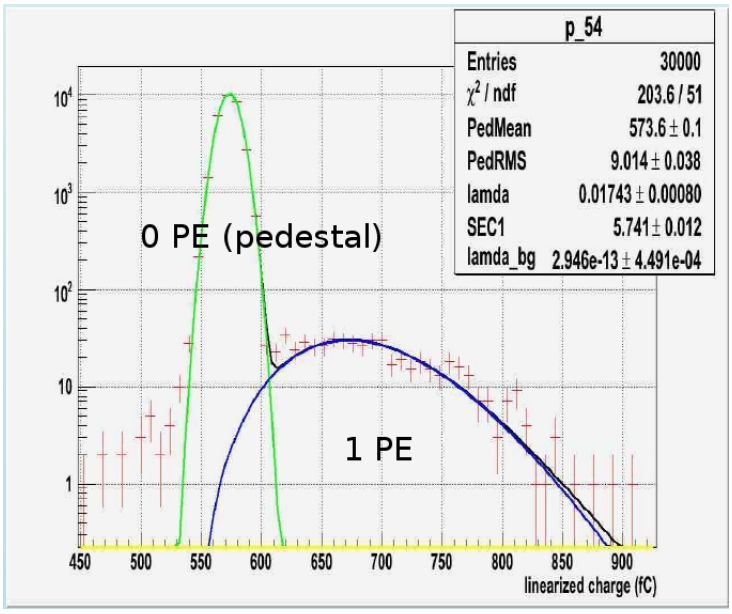

Figura 6.13: Resultados obtidos usando a distribução OnePE. Cortesia de Brandon Eberly, University of Pittsburgh

um segundo arquivo que contém os objetos de dados chamados *Tracks* e podem ser visualizados com o visualizador de eventos (*Event Display*). O resultado deste procesamento é mostrado na figura 6.14 para um evento de raios cósmicos.

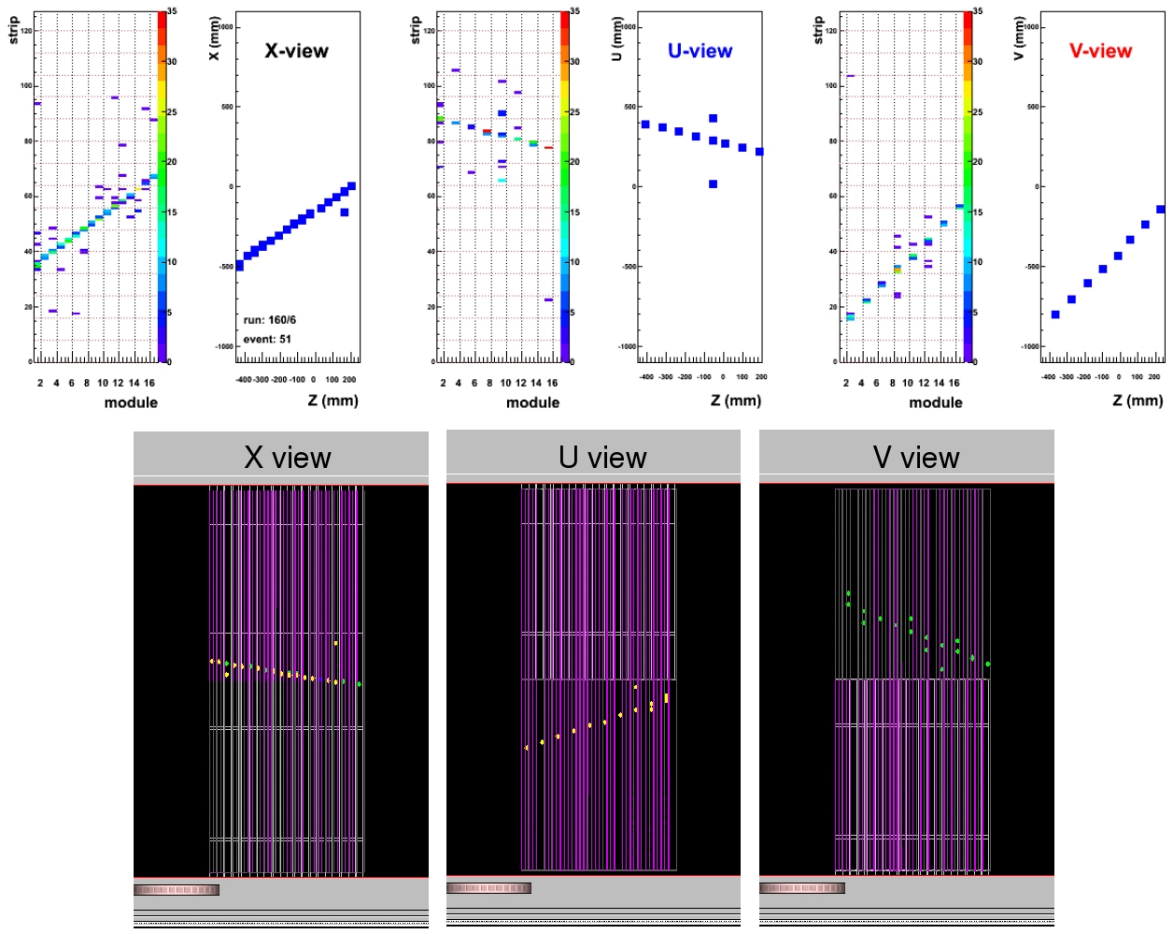

Figura 6.14: Resultado do processamento de runs Cosmics onde se mostra, na parte superior, as 3 vistas, XUV, visualizadas em ROOT. O lado esquerdo de cada vista mostra, simplesmente, as diferentes partes do detector que receberam luz (*hits*) e o lado direito mostra a trajetória do múon reconstruída como resultado do procesamento. Esta reconstrução empregou 12 módulos instalados no *TP*. A parte inferior mostra como estes eventos são visualizados pelo VISTA, o visualizador do MINERνA *software*.

Capítulo 7

Conclusões

Trabalhei no grupo de *software* do experimento MINERνA desenvolvendo, principalmente, a parte de processamento de dados e monitoramento em tempo real.

Foram adaptadas e desemvolvidas ferramentas baseadas no *software online* do LHCb para os sistemas *online* e quase-*online* que permitem o monitoramento em tempo real do experimento MINERνA . Embora ainda seja necessário mais trabalho, as bases já foram desenvolvidas.

O resultado de todo o trabalho se traduziu na criação do projeto *NEARONLINE* e a atualização do projeto *MINERVAONLINE* no *software* do MINERνA . Também criou-se o pacote *Histogramatron* que pertence ao projeto *NEARONLINE* onde serão desenvolvidos todos os algorítmos geradores de histogramas para o monitoramente em tempo real.

Procedimentos para usar ou administrar o sistema quase-*online* podem ser encontrados no manual que foi escrito para este fim e cuja versão em português é apresentada no apêndice D.

Trabalhei na montagem e comissionamento do Protótipo de Rastreamento do MINERνA (*Tracking Prototype*) que consiste em cerca de 20% do detector completo do MINERνA . Este protótipo foi, então , empregado para testar a operação de todos os sistema e partes do detector, tendo testado, com êxito:

- O projeto do detector.
- As fotomultiplicadores (PMTs).
- A eletrônica de na adquisição de dados.
- O sistema de injeção de luz.
- O *software* de adquisição de dados.
- O *software* de processamento em tempo real.
- O *software* de monitoramento em tempo real.

- O *software* de procesamento offline (reconstrução de trajetórias).

Os *software* desenvolvidos em meu trabalho foram empregados para a aquisição e monitoramento com o *Tracking Prototype* tendo-se, então , verificado e testado sua operação .

O experimento está completando o *Tracking Prototype* e preparando-se para instalá-lo no feixe de neutrinos NuMI. A tomada de dados de teste empregando neutrinos deve iniciar-se em abril de 2009, empregando os *software* por mim desenvolvidos.

Em continuação a este trabalho é preciso incorporar os *software* de reconstração e visualização de eventos ao *software* de processamento e monitoração em tempo real.

O experimento MINERνA entrará em operação completa no primeiro semestre de 2010 e deverátomar dados por cerca de quatro anos.

Apêndice A

Estrutura detalhada do *software* do MINERνA

Apresentamos aqui uma lista com a estrutura detalhada do *software* MINERνA versão v6r1 que entrou em vigência em febreiro de 2009. O *software* é estruturado em vários pacotes agrupados em um conjunto de projetos (mostrados na figura 4.4 do capitulo 4). Além de serem agrupados em projectos, que permite definir a dependência entre eles como mostrado na figura 4.4, a maioria dos pacotes posui prefixos do tipo Det/, Event/ e outros. Estes prefixos permitem agrupar os pacotes relacionados segundo sua funcionalidade. O significado de cada prefixo é dado a seguir:

- Det, pacotes que descrevem o detector (geometria, material etc.)
- Event, pacotes que definem o modelo do evento no experimento MINERνA .
- Kernel, pacotes que definem fundamentos e interfaces que se baseiam em outros pacotes.
- Sim, pacotes usados na simulação do detector.
- Gen, pacotes usados na geração de eventos.
- Vis, pacotes usados no visulizador de eventos.
- MinervaOnline, pacotes usados na aquisção de dados.

Estos prefixos são usados na maioria dos pacotes, mas não em todos eles como podese notar na lista completa de pacotes no experimento MINERνA aqui apresentada. Na estrutura de arquivos do *software* do MINERνA estes prefixos são traduzidos como pastas nos projetos dentro dos quais encontraremos os pacotes. Todos os pacotes podem ser encontrados no repositório CVS do experimento MINERνA :

`http://cdcvs0.fnal.gov/cgi-bin/public-cvs/cvsweb-public.cgi/?cvsroot=minervasw`

Nos repositórios CVS o uso de prefixos é muito conveniente já que eles permiten agrupar pacotes segundo seu prefixo independentemente dos projetos que a que pertençam. Segue a lista completa de pacotes no *software* do MINERνA.

- MINERVA
  - MinervaConf
  - MinervaPolicy
  - Kernel/MinervaKernel
  - Event/MinervaEvent
  - Event/MinervaEventBase
  - Event/MinervaGentEvent
  - Event/MinervaMCEvent
  - Det/MinervaDet
  - Det/IDDet
  - Det/ODDet
  - Det/MTDet
  - Det/MinervaDetDescChecks
  - Det/MinosMagnet
  - MinervaCondDBUI
  - Det/MinervaDDDBIF
  - Det/Plex
  - PedestalSuppression
- MDBASE
  - Det/MinervaDDDB
  - Det/MinervaSQLDDDB
  - MinosFieldmap
- EXTERNAL
  - PYTHIA6

  - GENIE
  - GDML
  - MINEXTERNAL
- WEBER
  - Sim/Weber
  - Sim/WeberAlgs
  - Sim/WeberMonitor
  - Sim/WeberPhysics
  - Sim/WeberTracker
- MEGA
  - Gen/CosmicGuns
  - Gen/Geration
  - Sim/GDMLwriter
  - Sim/Readout
  - Sim/Mega
- OWL
  - Event/MinervaTrackEvent
  - RawToDigit
  - Rec/TrackInterfaces
  - Rec/zTracker
- ENDEAVOR
  - DSTWriter
  - ForestTest
  - StripCal
  - SystemTests
- SOVIS
  - Vis/SoApp

  - Vis/SoWright

- VISTA
  - Vis/Vista

- MINERVAONLINE
  - MinervaOnline/CAEN2718
  - MinervaOnline/Readout
  - MinervaOnline/MinervaGaudiOnline
  - MinervaOnline/MinervaDAQ

- NEARONLINE
  - DaqRecv
  - Histogramatron

Apêndice B

Estrutura do Arquivo DST

Apresentamos aqui o conteúdo do arquivo DST, que é o resultado final do processamento de dados do experimento MINERνA .

Variável	Tipo	Descrição
ev_event	long int	número do evento na n-upla
ev_run	long int	número do run
ev_spill	long int	número da salva (*spill*)
ev_sub_run	long int	número do subrun
ev_gps_time_sec	long int	componente de segundos do tempo do evento pelo GPS
ev_gps_time_usec	long int	componente de micro-segundos do tempo do evento pelo GPS
ev_detector	long int	1-detector minerva, 2-protótipo de rastreamento, 3-m test
ev_det_config	long int	configuração do detector
ev_cal_settings	long int	ajuste de calibração
ev_readout	long int	configuração de leitura do DAQ
ev_trigger_type	long int	tipo de gatilho
ev_errors	long int	tipo de erro registrado pelo DAQ durante um evento

Tabela B.1: Informação acerca do evento no arquivo DST

Variável	Tipo	Descrição
n_rawhits	long int	número de *hits* brutos em um evento
n_idhits	long int	número de *hits* no *Inner Detector*(ID) em um evento
n_odhits	long int	número de *hits* no *Outer Detector*(OD) em um evento
n_clusters	long int	número de aglomerados (*clusters*) em um evento
n_tracks	long int	número de trajetórias em um evento

Tabela B.2: Parâmetros importantes de cada evento no arquivo DST

Variável	Tipo	Descrição
hits_total_pe	float	altura total do pulso de todos os *hits* em um evento em foto-elétrons
hits_id_per_mod	float	número de *hits* em um evento localizado no *Inner Detector*(ID) por módulo
hits_od_per_mod	float	número de *hits* em um evento localizado no *Outer Detector*(OD) por módulo

Tabela B.3: Informação acerca de conjuntos de *hits* no arquivo DST

Variável	Tipo	Descrição
hit_index	long int	índice para um *hit* em um evento
hit_time	float	tempo do *hit* em nanoseconds
hit_location	long int	1=*Inner Detector*(ID) , 2=*Outer Detector*(OD)
hit_module	long int	número do módulo de um *hit* no *Inner Detector*(ID)
hit_plane	long int	número do plano no módule (1-2)
hit_strip	long int	número da tira dentro do plano (1-127)
hit_frame	long int	número da estrutura de um *hit* no the *Outer Detector*(OD)
hit_tower	long int	seção na estrutura (1-6)
hit_story	long int	nível dentro de uma torre (1-4, 1=mais interno, 4=mais externo)
hit_bar	long int	barra cintiladora em uma camada (1-2)
hit_pe_cal	float	altura do pulso em foto-elétrons após calibração
hit_pe_raw	float	altura do pulso em foto-elétrons antes da calibração
hit_qhi	long int	altura do pulso em contagens de ADC na faixa alta do discriminador
hit_qlo	long int	altura do pulso em contagens de ADC na faixa baixa do discriminador
hit_qmed	long int	altura do pulso em contagens de ADC na faixa média do discriminador
hit_crate	long int	número do bastidor
hit_croc	long int	número do CROC
hit_chain	long int	canal do CROC (1-4)
hit_channel_id	long int	id do canal id
hit_board	long int	número do FEB na cadeia de CROCs (1-10)
hit_pixel	long int	número do elemento do PMT (0-63)
hit_num	long int	número do *hit*(0-7) durante o *gate* no elemento do PMT
hit_disc_fired	long int	se o discriminador disparou ou não
hit_sys_ticks	long int	tempo do disparo do discriminador em relação ao relógio do sistema

Tabela B.4: Informação acerca do um *hit* no arquivo DST

Variável	Tipo	Descrição
clus_index	long int	índice dos aglomerados em um evento
clus_size	long int	número de *hits* em u aglomerado
clus_hits_idx	long int	índice dos *hits* em um aglomerado
clus_pe	float	total de PE em um aglomerado
clus_time	float	tempo do maior *hit* hit (i.e *hit* com a pulso de maior altura) em um aglomerado
clus_view	long int	1=vista-x, 2=vista-u, 3=vista-v
clus_strip	long int	número da tira com o maior *hit* em um aglomerado
clus_coord	float	posição de um aglomerado ao longo da direção do plano (x,u,v)
clus_z	float	posição z de um aglomeradot no detector

Tabela B.5: Informação acerca do *cluster* no arquivo DST

Variável	Tipo	Descrição
trk_index	long int	índice para uma trajetória em um evento
trk_hits	long int	número de *hits* em uma trajetória
trk_nodes	long int	número de nós em uma trajetória
trk_node_cluster_idx	long int	índice para o aglomerado de um nó
trk_node_X	float	posição-x de um nó
trk_node_Y	float	posição-y de um nó
trk_node_Z	float	posição-z de um nó
trk_node_aX	float	inclinação do vetor de estado no plano X-Z que aponta para o próximo nó
trk_node_aY	float	inclinação do vetor de estado no plano Y-Z que aponta para o próximo nó
trk_node_qOverP	float	carga dividida pelo momentum de uma trajetória
trk_node_chi2	float	χ^2 de um nó
trk_chi2perDof	float	χ^2 reduzido de uma trajetória
trk_dof	long int	graus de liberdade de uma trajetória

Tabela B.6: Informação acerca de trajetórias no arquivo DST

Apêndice C

Software de Aquisição de Dados do MINERνA

(Original em inglês: Minerva's Data Acquisition Software)

C.1 Introdução

O *software* de aquisição de dados do MINERνA é dividido em duas partes: *software online* e *software* quase-*online* . O *software online* lê o *hardware* de aquisição de dados e monta uma sequência de dados chamada de *RawEvent* (um objeto de dados do GAUDI parte do modelo de evento do LHCb e que representa um único evento). Este *RawEvent* é armazenado em um disco rígido e enviado pela rede (transferência de evento) de forma a que a informação do evento seja disponibilizada para o processamento quase-*online* . O *software* quase-*online* recebe o *RawEvent* para processá-lo em um objeto de maior nível específico do Minerνa (no momento temos apenas *RawDigits* neste estágio) e cria histogramas baseados nestes novos objetos. Estes histogramas são publicados e podem ser requisitados, através da rede, por tarefas de monitoramento.

O *software online* e o *software* quase-*online* rodam em computadores dedicados conectados diretamente ao detector [1]. Eles se conectam por uma rede ethernet exclusiva local (LAN) onde o computador quase*online* serve como *gateway* visível fora da rede local, o que permite acesso remoto ao computador *online* para controle e monitoramento, conforme ilustrado na figura C.1.

Comunicação em rede (monitoramento, transferência de evento e controle de processo) são realizados empregando-se DIM, um sistema de comunicação para ambientes distribuidos/mistos. Para monitoramento o MINERνA emprega o GAUCHO, que é uma implementação de DIM no LHCb que permite que processos *online* baseados em GAUDI

[1] Em janeiro de 2009 estes dois computadores estão localizados no laboratório WideBand do FERMILAB, onde encontra-se o protótipo de rastreamento, e são identificados como MinervaTPDAQ (*online*) e minervadaq02 (quase-*online*).

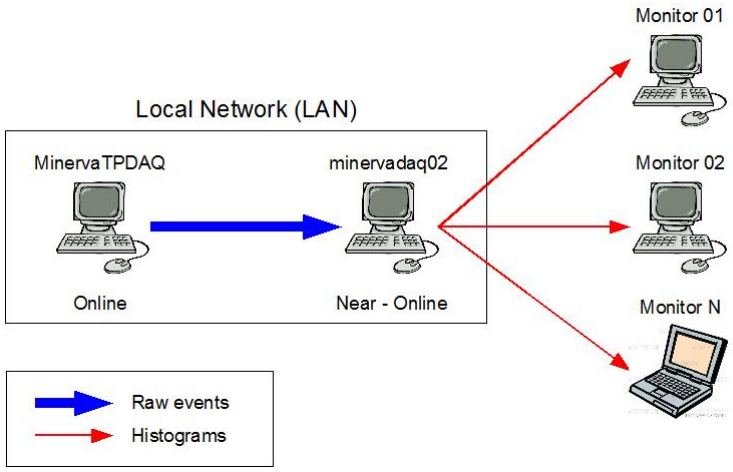

Figura C.1: Montagem para aquisição de dados montada no laboratório WideBand do FERMILAB em janeiro de 2009

publiquem histogramas. O controle de processo é uma característica dos dos processos *online* baseados em GAUDI, cujo conjunto de estdos possíveis (estados FSM) pode ser controlado e monitorado remotamente. A transferência de eventos também é baseada em uma implementação DIM do LHCb. Esta implementação emprega DIM para transferir objetos *RawEvent* entre processos rodando em máquinas diferentes.

O MINERνA está desenvolvendo seu software de processamento quase-*online* e de aquisição de dados baseado no *software online* do LHCb. O processamento quase-*online* é o *software* que fornecerá histogramas para o monitoramento *online* do MINERνA . Algumas das ferramentas e componentes principais do *software online* do LHCb estão descritas na seção C.2. A seção C.3 explica como o MINERνA os implementa em seu próprio *software* quase-*online* e em seu sistema de aquisição de dados.

C.2 Modelo de *software online* do LHCb

Esta seção descreve o *software online* do LHCb. Todos os pacotes e códigos do *software* são parte do projeto ONLINE localizado em

```
${SITEROOT}/lhcb/ONLINE/ONLINE_v*r*/
```

O projeto ONLINE emprega e depende do projeto LHCb. Todos os objetos do Modelo de Evento do LHCb são definidos em

```
${SITEROOT}/lhcb/LHCB/LHCB_v*r*/
```

C.2.1 DIM

O DIM (Gerenciamento de Informação Distribuída - *Distributed Information Management*) é um pacote leve de comunicação entre processos baseado no paradigma cliente/servidor. Servidores são definidos como processos que tem alguma informação para bublicar (um serviço) e Clientes são os processos que precisam desta informação . DIM é o principal pacote para comunicação de rede no modelo de *software* do LHCb. A figura C.2 mostra os componentes do DIM.

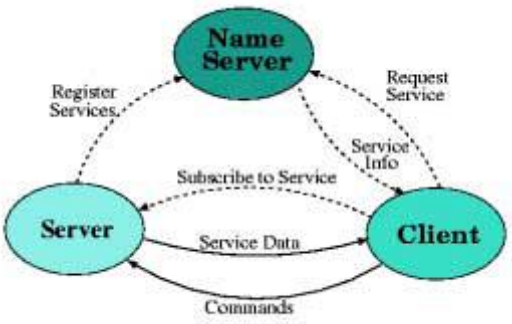

Figura C.2: Diagrama de fluxo de dados do DIM

Todos os servidores DIM registram-se no Servidor de Nomes (Servidor de Nomes do DIM ou DNS) e registram todos os seus serviços. O DNS armazena uma lista com todos os servidores que tenham se registrado e os serviços que cada um deles oferece. Quando um client DIM começa ele solicita um serviço ao DNS e recebe a localização (endereço IP) do servidor que oferece este serviço. O Cliente, então, registra-se diretamente ao servidor e a comunicação tem início. isto significa:

- O Cliente pode receber informação atualizada em intervalos regulares.
- O Cliente pode receber informação atualizada sempre que ocorrer alguma alteração.
- O Cliente pode enviar comandos ao servidor.

Os três elementos básicosThe three basic components of DIM (mostrados na figura C.2) podem rodar em máquinas diferentes (ou na mesma máquina para testes) e podemos ter varios deles. Clientes e servidores sabem a localização (endereço IP) onde o DNS está rodando por intermédio da variável de ambiente DIM_DNS_NODE. A principal biblioteca do DIM (dim.dll em Windows ou libdim.so em Linux) define todas as classes necessárias para a implementação de clientes ou servidores DIM em uma aplicação . O pacote DIM pode ser encontrado dentro do *software* do LHCb em

```
${SITEROOT}/lhcb/ONLINE/ONLINE_v*r*/Online/DIM/v*r*/
```

Há exemplos de como usar estas classes em

```
http://dim.web.cern.ch/dim/dim_user.html
```

Os serviços publicados pelos servidores podem ser um tipo de dado simple (integer, float, double, short, longlong, char) ou, uma vez que o tamanho seja especificado, typo de dado complexo consistindo de um conjunto misto de tipos dee dado simples. Há um tipo especial de serviço chamado de comando DIM empregado pelos servidores para receber comandos dos clientes. O nome do serviço é livre (DIM não coloca restrições na forma dos nomes dos serviços) e cabe ao usuário a definição da convenção para os nomes.

O pacote DIM contém, também, vários aplicativos já compilados prontos para serem usados. Entre os mais úteis citamos:

dns.exe Dá partida ao servidor DNS DIM.

did.exe (DID-Distributed Information Display) inicia um GUI simples que permite a monitoração de qualquer DNS que esteja rodando. Permite que o usuário veja todos os servidores registrados no DNS, a lista dos serviços por servidor, ler os serviços dos servidores e enviar comandos.

dim_send_command.exe é uma linha de comando que permite ao usuário enviar um comando a um servidor. Emprego:

```
dim_send_command.exe NameOfCommandService command
```

dim_get_service.exe é uma linha de comando que permite o usuário ler o conteúdo de um serviço publicado. Emprego:

```
dim_get_service.exe NameOfService
```

Com excessão do primeiro, todos estes aplicativos precisam saber onde o DNS está rodando. Como qualquer cliente DIM, Eles se utilizam da variável DIM_DNS_NODE para etse fim.

Embora qualquer sequência de caracteres possa ser usada como NameOfCommandService, é uma boa idéia atribuir a ele o nome do servidor.

C.2.2 GaudiOnline

A estrutura do GAUDI, onde algoritmos nunca se comunicam diretamente com a armazenagem permanente de dados, a torna adequada para aplicativos *online* onde os dados vem do DAQ. Apenas o Serviço de Entrada da Armazenagem Transiente, o controle de programa e os componentes de monitoramento precisam ser especializados para se interfacearem com o DAQ e com o sistema de controle, enquanto outros componentes podem ser usados da forma como em aplicativos *offline*.

Estas especializações são feitas no pacote GaudiOnline:

```
${SITEROOT}/lhcb/ONLINE/ONLINE_v*r*/Online/GaudiOnline/v*r*/
```

Este pacote contain o código para o principal aplicativo no ambiente *online* : Gaudi.exe. É usado de forma similar a forma como o código de mesmo nome é usado no ambiente *offline* para rodar qualquer programa GAUDI. Este pacote contém também o código para a principal biblioteca do ambiente *online* que define todas as especializaçõs GAUDI mencionadas anteriormente: libGaudiOnline.so ou GaudiOnline.dll.

Para rodar um programa GAUDI online precisamos digitar algo do tipo:

```
Gaudi.exe GaudiOnline OnlineTask -opts=JobOptions.opts
```

O aplicativo principal neste comando (Gaudi.exe) carrega a biblioteca principal (GaudiOnline) e chama a função OnlineTask definida na biblioteca. A função OnlineTask inicia um Gaudi-ApplicationManager com o arquivo de opções especificado em JobOptions.opts. Em adição à função OnlineTask podemos chamar outras funções que são definidas no arquivo DllCalls.cpp dentro do pacote GaudiOnline. Podemos, por exemplo, chamar a função help_OnlineTask para obter uma ajuda mais detalhada sobre como usar a função OnlineTask:

```
Gaudi.exe GaudiOnline help_OnlineTask
```

O principal aplicativo Gaudi.exe é muito geral. Ele pode carregar qualquer biblioteca e chamar qualquer função da biblioteca. Uma biblioteca muito útil é a biblioteca OnlineKernel definida em

```
${SITEROOT}/lhcb/ONLINE/ONLINE_v*r*/Online/OnlineKernel/v*r*/
```

O aplicativo principal pode ser usado da seguinte maneira:

```
Gaudi.exe OnlineKernel mbm_mon
```

onde mbm_mon é uma de várias funções definidas na biblioteca OnlineKernel. Esta função é explicada na seção C.2.4. Embora o principal aplicativo, Gaudi.exe, seja usado, este não é um programa GAUDI *online* (não há qualquer gerente de aplicativo nele).

Vários exemplos em como o LHCb usa o aplicativo principal podem ser encontrados em:

```
${SITEROOT}/lhcb/ONLINE/ONLINE_v*r*/Online/GaudiOnline/v*r*/tests/
```

Quando um programa *online* GAUDI roda ele cria um servidor DIM cujo nome é dado pelo valor da variável de ambiente UTGID (User assigned unique Thread Group Identifier). Este nome não é, necessariamente, o nome do processo. Cada um destes servidores DIM publica um serviço de comando (explicado na seção C.2.1) usando o próprio nome do servidor (UTGID) de forma que eles podem receber comandos (config, start, stop, reset and unload) que controlam o Gaudi-ApplicationManager dentro dele. Eles também publicam dois serviços adicionais chamados UTGID/status e UTGID/fsm_status que fornecem o estado do processo (not_ready, ready, stopped or runnning) e o estado da transição entre estados (transition complete, transition failed, executing transition) respectivamente. A documentação do LHCb refere-se a eles como estados FSM e eles são definidos nos arquivos DimTaskFSM.cpp e GaudiDimFSM.cpp do pacote GaudiOnline. A figura C.3 mostra todos estes possíveis estados e transições.

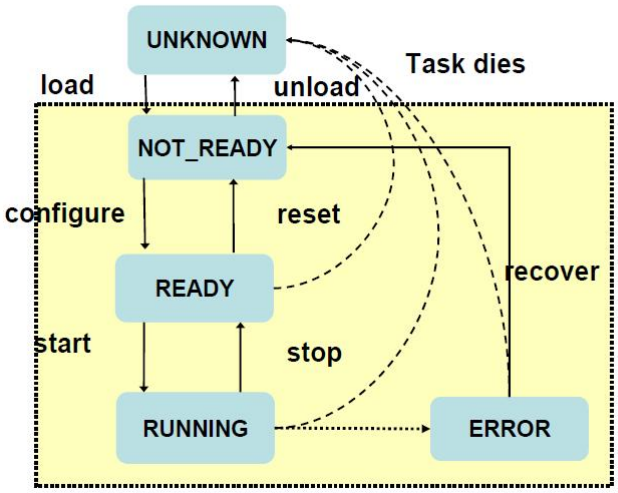

Figura C.3: Diagrama de estados dos processos rodando como programas GAUDI *online* .

Para iniciar um programa GAUDI *online* especificando seu UTGID (caso senhum UTGID seja especificado o número PID do processo será usado) devemos digitar:

```
set UTGID=TestServer
Gaudi.exe GaudiOnline OnlineTask -opts=JobOptions.opts
```

Para controlar e monitorar este programa GAUDI *online* podemos empregar o aplicativo do DIM como, por exemplo:

- Para enviar um comando:

```
dim_send_command.exe TestServer configure
```

- Para conhecer o estado da transição :

```
dim_get_service.exe TestServer/fsm_status
```

- Para conhecer o estado do processo:

```
dim_get_service.exe TestServer/status
```

Podemos também usar a ferramenta DID para o mesmo objetivo. Uma vez iniciada, seu uso é muito fácil e intuitivo.

C.2.3 GAUCHO

O GAUCHO (GAUdi Component Helping Online) é o pacote que permite que algorítimos em um programa GAUDI *online* publique variáveis e histogramas. Este pacote pode ser encontrado em

```
${SITEROOT}/lhcb/ONLINE/ONLINE_v*r*/Online/Gaucho/v*r*/
```

O GAUCHO define o serviço MonitorSvc que é uma implementação da interface GAUDI IMonitorSvc. MonitorSvc torna possível a publicação de variáveis internas (bool, integer, long, double, string, par de doubles e arrays) criadas como serviços DIM através do método declareInfo. Para publicar, por exemplo, um inteiroprecisamos adicionar as seguintes linhas no no initialize do algorítm DAUDI:

```
declareInfo("counter",m_counter,"Counter value");
```

As linhas seguintes também precisam estar presente no arquivo de opções do programa GAUDI:

```
ApplicationMgr.DLLs   += { "Gaucho" };
ApplicationMgr.ExtSvc += { "MonitorSvc" };
ApplicationMgr.TopAlg  = { "MyAlgorithm/TestAlg" };
```

Isto cria um serviço DIM chamado UTGID/TestAlg/counter. Este serviço conterá qualquer que seja o valor que a variável m_counter tenha no momento que o serviço é solicitado.

O MonitorSvc também permite a publicação dinâmica de histogramas encontrados na armazenagem transiente. Isto significa que o usuário não tem que chamar o mťodo declareInfo para histogramas. O MonitorSvc realiza isto através da classe DimCmdServer encontrada no pacote GAUCHO. Graças à esta propriedade, basta declarar o histograma no executes do algorítimo GAUDI pela adiçõ de uma linha do tipo:

```
plot1D(qhi,"charge",0,3500,70);
```

Um serviço DIM chamado UTGID/TestAlg/charge será publicado e conterá toda a informação dos histogramas como um array de inteiros. Um cliente chamado GaudiMonitor.exe (no ROOT) e DIM (encontrado no pacote HPDMonitor) pode criar uma visualização deste histogramas. O cliente é capaz de entender o array e transformá-lo em um histograma do ROOT através da classe DimInfoHisto (encontrada no pacote HPDMonitor). Este pacote pode ser encontrado em:

```
${SITEROOT}/lhcb/ONLINE/ONLINE_v*r*/Online/HPDMonitor/v*r*/
```

Há, também, outro cliente para histogramas chamado Presenter, encontrado em

```
${SITEROOT}/lhcb/ONLINE/ONLINE_v*r*/Online/Presenter/v*r*/
```

Este cliente é mais complexo e parece ser mais completo. Entretanto, ainda não foi testado pelo experimento MINERνA .

O MonitorSvc também inicia um DimPropertyServer. Esta classe é capaz de usar DIM para obter ou fornecer , remotamente, os valores das propriedades de programas *online* GAUDI (as propriedades especificadas em seu arquivo de opções). Não foi ainda testado no experimento MINERνA .

C.2.4 Modelo de processamento de dados de HLT do LHCb

LHCb emprega 2.000 CPU em seu cluster HTL (Gatilho de Alto Nível - High Level trigger) Cada nó tem algorítmos de seleção que rodam em uma arquitetura de processamento de dados baseada em buffers de eventos de memória compartil;hada.

Esta arquitetura é feita com programas *online* GAUDI (ou simplesmente "tasks") chamados *Producers* e *Consumers* que se comunicam por meio de uma área de memória compartilhada gerenciada: o gerente de buffer (BM). A figura C.4 mostra um bloco de construção da arquitetura. Os dados que estes programas ("tasks") podem manipular são objetos GAUDI de dados chamados *RawEvents* definidos no *software* do LHCb. *Producers* são os programas

("tasks") que declaram *RawEvents* no BM e *Consumers* são os que lêem e fornecem *RawEvents* do BM. O BM é gerenciado por um programa *online* GAUDI que roda em cada nó, que cria e inicializa a área de memória compartilhada.

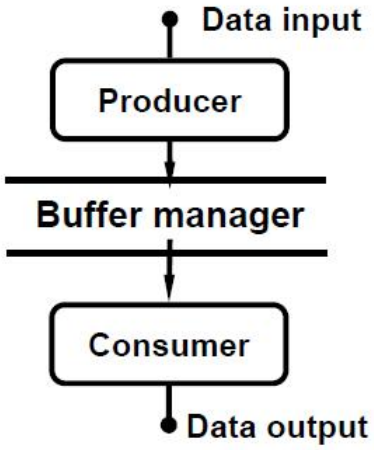

Figura C.4: Bloco de construção básico para processamento de dados HLT do LHVb.

Cada tarefa empregando o BM registra-se com um nome (que não é necessariamente o nome do servidor DIM criado pela tarefa) e um identificador de partição (PartitionID). *Consumers* receberão apenas dados declarados por *producers* com o mesmo identificador de partição e com uma máscara booleana de aceitação ou veto de 128 bits, que permite que eles sejam específicos quanto ao tipo de dados the querem receber. *Consumers* podem solicitar dados ao BM de três maneiras diferentes:

- O *consumer* vê todos os dados declarados ao BM segundo a máscara de pedidos.
- Um grupo de *consumers* vê os dados exatamente uma vez. Este modo permite a execução múltipla de um aplicativo aproveitando-se de CPU the múltiplos núcleos.
- Um *consumer* é atendido na base do melhor esforço. Neste modo *producers* são sempre capazes de declarar novos dados. Dados ainda pendentes , que precisem ser liberados para atender a exigências de espaço, não serão vistos por estes *consumers*.

As propriedades dos *consumers* são especificadas em seu respectivos arquivos de opções. A relação entre o valor no arquivo de opções e estas propriedades ainda precisa ser definida. As linhas no arquivo de opções que especificam estas propriedades são da forma seguinte:

```
EventSelector.REQ1 = "EvType=2;            //(1, 2, 3)
                     TriggerMask=0xffffffff,0xffffffff,0xffffffff,0xffffffff;
                     VetoMask=0,0,0,0;
```

```
MaskType=ANY;        //(ANY/ALL)
UserType=VIP;        //(USER/VIP/ONE)
Frequency=PERC;      //(MANY/PERC)
Perc=100.0";
```

O código definindo *Producers*, *Consumers* e *BM* pode ser encontrado na biblioteca OnlineKernel definida no pacote:

```
${SITEROOT}/lhcb/ONLINE/ONLINE_v*r*/Online/OnlineKernel/
```

A biblioteca OnlineKernel oferece várias funções úteis. Uma das mais importantes é *mbm_mon* que permite a monitoração de qualquer MB local. Ela pode mostrar, por exemplo, o nome do BM, todos os *Producers* e *Consumers* conectados aoo número de eventos produzidos, eventos consumidos para cada *Consumer*. Não é um programa *online* GAUDI mas pode ser lançado pelo aplicativo Gaudi.exe da seguinte maneira (ver seção C.2.2):

```
Gaudi.exe OnlineKernel mbm_mon
```

A arquitetura tem dois tipos adicionais de blocos de construção:

Unidade Reformatadora de Dados: construído conectando a entrada de um *producer* a outro gerenciador de buffer (lado esquerdo da figura C.5). O processo de reformatação pega dados em um buffer de entrada e então os transforma ou copia e declara o resultado em um buffer de saída. Nem todos os eventos recebidos recebidos do buffer de entrada precisam, necessariamente, ser declarados no buffer de saída. Assim, esta unidade é ideal para filtragem de eventos. No momento em que os processos reformatadores declaram os eventos aceitos eles também especificam a máscara de gatilho que permitirá, mais tarde, que as tarefas consumidoras (*consumers*) classifiquem os eventos sem analizar seus conteúdos de dados.

Unidade de Transferência de Dados: construída conectando a saída de um *consumer* à entrada de um *producer* em um nó remoto usando uma conexão de rede (lado direito da figura C.5). É usada para mover os eventos aceitos do cluster HLT para o sistema de armazenamento. ESta é a unidade que realiza a transferência de eventos.

O LHCb construiu seu cluster HLT empregando estes blocos de construção. Em cada nó deste cluster há um conhubto de programas *online* GAUDI (*producers*, *consumers* e BM) como mostrado na figura C.6:

Os cartões de leitura enviam blocos de dados contendo fragmentos de múltiplos eventos, chamados de Pacotes Multi-Eventos (MEP), para os processos de Construção de eventos (EB)

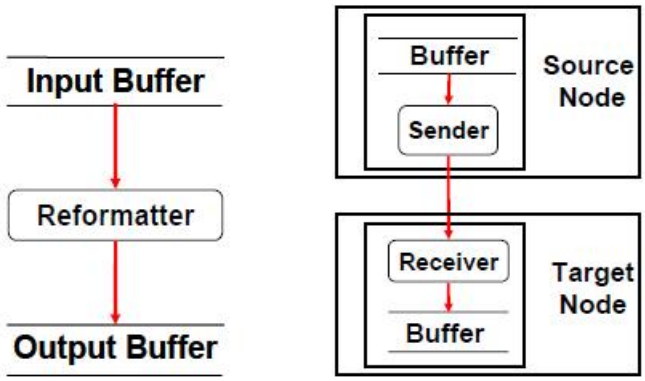

Figura C.5: Unidades reformatadoras de dados e unidades de transferência de dados capazes de transferir dados entre gerentes de buffers em dois nós.

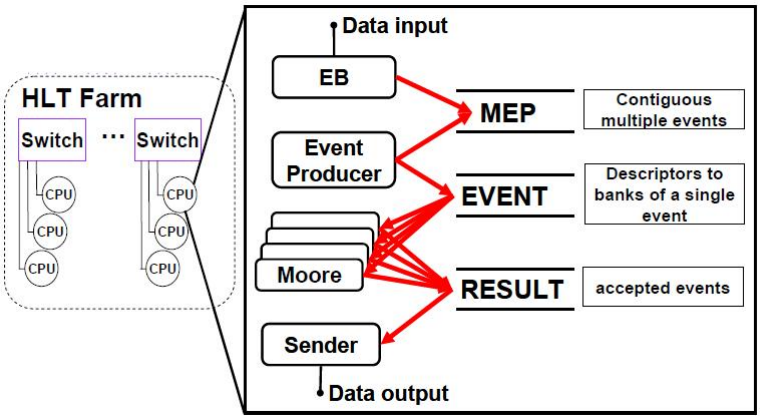

Figura C.6: Os diferentes fluxos de dados de eventos através de diferentes gerenciadores de buffer entre as tarefas em execução em um nó de cluster HLT.

rodando em cada nó. O EB monta os fragmentos em um bloco contínuo, um Multi Evento, e deposita os eventos no BE "MEP". O Produtor de Eventos, registrado no BM "MEP", analiza o formato dos dados e constroi *RawEvents*, que são declarados ao BM "EVENT". Os processos de filtragem de eventos (Moore) acessam os dados do evento e decidem se o evento deve ser mantido. Quando a decisão é positiva os *RawEvents* são copiados para o buffer "RESULT"onde o processo Sender os lê e os envia para o sistema de armazenagem.

Consumers e *Producers* são programas *online* GAUDI que podem ser controlados e monitorados com DIM conforme explicado na seção C.2.2. Se algum *Consumers* executa algorítimos que criam histogramas, eles serão publicados usando o MonitorSvc do GAUCHO conforme explicado na seção C.2.3.

Há vários exemplos de *scripts* que lançam todos estes processos e exemplos dos arquivos de opções GAUDI necessários. Eles podem ser encontrados em:

```
${SITEROOT}/lhcb/ONLINE/ONLINE_v*r*/Online/OnlineTasks/
```

C.3 Modelo do *software online* do MINERνA

C.3.1 *Software online*

O *software online* precisa rodar em uma máquina conectada diretamente ao *hardware* do detector e que com o sistema operacional WINDOWS (ver nota de rodapé na seção C.1). Ele consiste de diversos pacotes. Os pacotes específicos do MINERνA estão no projeto MINERVAONLINE que usa e depende unicamente do projeto ONLINE do LHCb. Este projeto foi feito para ser compilado no sistema operacional WINDOWS e está localizado em

```
${SITEROOT}/minerva/MINERVAONLINE/MINERVAONLINE_v*r*/
```

O projeto MINERVAONLINE consiste dos seguintes pacotes:

- CAEN2718
- Readout
- TestStandHV
- MinervaGaudiOnline
- MinervaDAQ

Os três primeiros pacotes contêm todas as biliotecas necessárias para comunicação com o *hardware* de aquisição de dados (DAQ). MinervaGaudiOnline contém uma implementação da classe *Producer* específica para o MINERνA chamada *MinervaProducer*. MinervaDAQ contém o aplicativo principal (MinervaDAQMain.exe) que conecta os drivers do *hardware* do DAQ com o GaudiOnline do LHCb.

Pacotes adicionais empregados pelo *software online* do MINERνA pertencem ao LHCb. São eles que contêm, basicamente, o código para os *Producers*, *Consumers Buffer Managers* que podemos encontrar em

```
${SITEROOT}/lhcb/ONLINE/ONLINE_v*r*/Online/OnlineKernel
```

and the ones that contains the code for the main online application and library

```
${SITEROOT}/lhcb/ONLINE/ONLINE_v*r*/Online/GaudiOnline
```

Para rodar nosso detector nós usamos o *script* I runminervadaq.bat que pode ser encontrado em

```
set UTGID=MBMMon
start "%UTGID%" %GAUDIONLINEROOT%\%CMTCONFIG%\Gaudi.exe OnlineKernel.dll mbm_mon
```

Este não é um programa *online* GAUDI e pertence inteiramente ao ONLINE do LHCb.

OutputBuffer

Este processo cria um Gerenciador de Buffer (BM) chamado "OUT". Ele é chamado pelo *script* runminervadaq.bat da seguinte maneira:

```
set UTGID=OutputBuffer
start "%UTGID%" %gaudi_exe% -main=%OPTS%\MBMinit.opts
                            -opt=%OPTS%\Daemon.opts
                            -msgsvc=%msg_svc%
```

A variável gaudi_exe é definida como

```
gaudi_exe=
%GAUDIONLINEROOT%\%CMTCONFIG%\Gaudi.exe GaudiOnline.dll OnlineTask -auto
```

Este processo é um programa *online* GAUDI com UTGID=OutputBuffer. Ele utiliza o arquivo de opções MBMinit.opts encontrado em

```
{SITEROOT}/lhcb/ONLINE/ONLINE_v*r*/Online/GaudiOnline/options
```

As linhas mais importantes neste arquivo de opções são

```
#include "$GAUDIONLINEROOT/options/OnlineEnvironment.opts"
ApplicationMgr.ExtSvc        += { "LHCb::MEPManager/OutputBuffer" };
OutputBuffer.InitFlags        = "-s=8096 -e=64 -u=64 -i=OUT -c";
OutputBuffer.PartitionBuffers = true;
OutputBuffer.PartitionID      = @OnlineEnv.PartitionID;
```

InitFlags define o tamanho em kbits(s=8096), número m,áximo de eventos (e=64), número máximo de *producers* ou *consumers* (u=64) e o nome (i=OUT) do buffer. MINERνA usa 14d (notação hexagesimal para 333) como partitionID em todos seu sistema de aquisição *online* . Este é o valor padrão no código do LHCb e é especificado na propriedade PartitionID (a variável @OnlineEnv.PartitionID é definida no arquivo de opções OnlineEnviroment.opts). Quando marcadoco co verdadeiro, PartitionBuffers simplesmente anexará o PartitionID ao nome do buffer de forma que o novo nome deste buffer será OUT_14d.

Este processo também pertence inteiramente ao ONLINE do LHCb. Usa um arquivo de opções do LHCb.

DiskWR

Este processo é uma tarefa *consumer* com UTGID=DiskWR. Ela se registra no OUT_14d BM com seu UTGID (DiskWR) e a identidade de partição 14d. Ela aguardará a declaração de *RawEvents* pelo BM. Este processo é chamado pelo *script* runminervadaq.bat da seguinte forma:

```
set UTGID=DiskWR
start "%UTGID%" %gaudi_exe%
   -main=%OPTS%\Main.opts
   -opt=%MINERVAGAUDIONLINEROOT%\options\MinervaDiskWR.opts
   -msgsvc=%msg_svc%
```

Este é um programa tipo *consumer* GAUDI *online* que é lançado com um arquivo de opções especifico do MINERνA localizado no pacote MinervaGaudiOnline. Este arquivo de opções contém as seguintes linhas:

```
ApplicationMgr.ExtSvc      += { "LHCb::MEPManager/MEPManager" };
MEPManager.Buffers          = { "OUT" };
MEPManager.PartitionID      = 333;
MEPManager.PartitionBuffers = true;
```

Estas linhas especificam o nome do BM ao qual este *consumer* deve se conectar ("OUT"). Especifica, também, a identidade da partição (333 ou 14d). A propriedade PatitionBuffers, da mesma forma que no caso do OutputBuffer, simplesmente indica que o nome do buffer tem que conter a identidade da partição (OUT_14d).

O arquivo de opções também especifica o seguinte:

```
EventSelector.Input  = "OUT";
EventSelector.Decode = false;
EventSelector.REQ1   = "EvType=2;
                       TriggerMask=0xffffffff,0xffffffff,0xffffffff,0xffffffff;
                       VetoMask=0,0,0,0;
                       MaskType=ANY;
                       UserType=VIP;
                       Frequency=PERC;
                       Perc=100.0";
```

A propriedade Input do EventSelector indicai de qual buffer a tarefa deve ler o *RawEvents* (em nosso caso, de apenas um: OUT). A propriedade REQ1 especifica o tipo de evento que nossa tarefa deve consumir, como explicado na seção C.2.4. Temos, aqui, todos os valores padrão que permitirão nossa tarefa consumir todos os *RawEvents* declarados no BM.

O arquivo de opções também especifica o seguinte:

```
ApplicationMgr.OutStream    = { "RawEvent" };
RawEvent.ItemList           = { "/Event#1"
                              , "/Event/DAQ#1"
                              , "/Event/DAQ/RawEvent#1"};

OnlineEnv.RawEventsFile     =
"DATAFILE='PFN:RawData.root' TYP='POOL_ROOTTREE' OPT='RECREATE'";

#include "./Output.opts"
RawEvent.Output             = @OnlineEnv.RawEventsFile;
```

Estas linhas são bem conhecidas no ambiente *offline* GAUDI. Elas especificam os elementos da Armazenagem Transitória de Eventos (*Trasient Event Store*) que queremos escrever em disco. *RawEvents* é escrito em um arquivo cujo nome e formato são especificados em *RawEvent.Output* e que são iguais ao valor da variável *OnlineEnv.RawEventsFile*. O valor padrão é um arquivo ROOT em formato POOL chamado RawData.root. Esta variável pode ser redefinida no arquivo incluído Output.opts com um nome que depende de várias variáveis (data, número da tomada de dados, etc). O arquivo Output.opts é criado no *script* runminervadaq.bat em uma linha como a seguinte:

```
echo OnlineEnv.RawEventsFile =
"DATAFILE='PFN:%3_RawEvents.inprogress' TYP='POOL_ROOTTREE' OPT='RECREATE'";
>  Output.opts
```

Deve-se notar que não há um algorítmo GAUDI rodando dentro desta tarefa. Tudo que ela faz é permitir que os serviços GAUDI apropriados encham a armazenagem transiente de eventos GAUDI com *RawEvents* e, então , escreve-os em disco.

NetSend

Este processo é uma tarefa *consumer* UTGID=NetSend. Ele registra-se no BM OUT_14d BM com seu UTGID (NetSend) e a identidade de partição 14d. Aguardará que *RawEvents* seja declarado no BM. Este processo é chamado pelo *script* runminervadaq.bat da seguinte maneira:

```
set UTGID=NetSend
start "%UTGID%" %gaudi_exe%
   -main=%OPTS%\Main.opts
   -opt=%MINERVAGAUDIONLINEROOT%\options\MinervaNetSend.opts
   -msgsvc=%msg_svc%
```

O arquivo de opções para este programa GAUDI tipo *online consumer* é muito similar ao do programa DiskWR job. Uma das diferenças é que ele não contém as linhas que especificam que objetos da Armazenagem Transiente de Eventos devem ser escritos em disco. Outra diferença é que o programa NetSend contém o seguinte algorítmo GAUDI:

```
ApplicationMgr.TopAlg  += {"LHCb::DimDataSender/Sender"};

Sender.DataSink         = "NetRecv";
Sender.UseEventRequests = false;
Sender.AllowSuspend     = true;
Sender.Connection       = "";
Sender.Compress         = 0;
Sender.ChecksumType     = 1;
Sender.GenerateMD5      = false;
Sender.DataType         = 2; // MDF_NONE=1, MDF_RECORDS=2, MDF_BANKS=3
Sender.BankLocation     = "LHCb::RawEventLocation::Default";
```

Este algorítmo GAUDI (DimDataSender) é o que realiza a transferência de eventos. Ele consome *RawEvents* do buffer OUT e usa DIM para enviá-los para um processo remoto cuja UTGID é especificada na propriedade *Sender.DataSink*. Este programa é um *Consumer* no sentido de que *consome* eventos do buffer *OUT* mas é também chamado de *Produtor de Rede (Network Producer)* pois ele *produz* eventos para a rede.

MinervaMainDAQ

Quando iniciado, MinervaDAQMain.exe lê dados dos drivers do *hardware* do DAQ, monta um eventov (Event Building) e cria a sequência de evento chamada *RawEvent* de forma similar ao "Event Producer" do LHCb (ver figura C.6) usando, entretanto, a classe *MinervaProducer* ao invés da *Producer*. *MinervaProducer* registra-se no buffer OUT_14d com o nome PROD_0 e a mesma identificação de partição usada pelos *Consumers*. Este não é um programa GAUDI *online* pois não executa um servidor DIM e não tem uma UTGID definida. Este processo não pode ser monitorado nem controlado por meio de DIM.

O projeto NEARONLINE consiste de:

- DaqRecv

DaqRecv é um pacote muito simples contendo apenas *scripts* e arquivos de opções. Os *scritps* basicamente executam programas GAUDI *online* do GaudiOnline do LHCb mas com arquivos de opções específicos do MINERνA definido em DaqRecv.

Há três processos que rodam lançados pelo *script* daq_recv.sh na seguinte sequência:

- DNS.exe
- Gaudi.exe (UTGID=InputBuffer)
- Gaudi.exe (UTGID=NetRecv)
- Gaudi.exe (UTGID=Reformatter)

DNS.exe

Este processo lança um servidor DNS (ver seção C.2.1). O DNS precisa rodar em uma máquina quase-*online* porque ele precisa ser visto pelos monitores clientes (ver figura C.1).

Inputbuffer

Este processo é muito similar ao OutputBuffer usado no *software online* . Ele lança um BM chamadoa "Event". Ele recebe chamadas do *script* daq_recv.sh script da seguinte maneira:

```
export UTGID=InputBuffer
${GAUDIONLINEROOT}/${CMTCONFIG}/Gaudi.exe
${GAUDIONLINEROOT}/${CMTCONFIG}/libGaudiOnline
OnlineTask
-auto
-main=${GAUDIONLINEROOT}/options/MBMfarm.opts
-opt=${GAUDIONLINEROOT}/options/Daemon.opts
-msgsvc=MessageSvc
```

Este é um programa *online* GAUDI com UTGID=InputBuffer. Ele usa o arquivo de opções MBMfarm.opts localizado no pacote GaudiOnline. Este arquivo de opções especifica o seguinte:

```
#include "$GAUDIONLINEROOT/options/OnlineEnvironment.opts"
ApplicationMgr.ExtSvc        += { "LHCb::MEPManager/OutputBuffer" };
OutputBuffer.InitFlags        = "-s=8096 -e=64 -u=64 -i=Event -c";
```

```
//OutputBuffer.PartitionBuffers = true;
OutputBuffer.PartitionID      = @OnlineEnv.PartitionID;
```

Este é, basicamente, o mesmo arquivi de opções usado pelo OutputBuffer. As diferenças são o nome do buffer (neste caso "Event") e o valor da propriedade PartitionBuffers (o valor padrão neste caso é "false"). Isto significa, apenas, que o nome do buffer não conterá o identificador de partição .

NetRecv

Este processo é uma tarefa tipo *producer* BM que registra-se no buffer "Event"com seu UTGID (NetRecv) e com identificador de partição 14d. Este processo é chamado pelo *script* daq_recv.sh da seguinte maneira:

```
export UTGID=NetRecv;
${GAUDIONLINEROOT}/${CMTCONFIG}/Gaudi.exe
${GAUDIONLINEROOT}/${CMTCONFIG}/libGaudiOnline
OnlineTask
-auto
-main=${GAUDIONLINEROOT}/options/Main.opts
-opt=${MINERVAGAUDIONLINEROOT}/options/MinervaNetRecv.opts
-msgsvc=MessageSvc
```

Este é um programa *online* GAUDI com UTGID=NetRecv. É também chamado *Consumidor de Rede (Network Consumer)* porque recebe *RawEvents* do *Produtor de Rede (Network Producer)* no nó *online* e os declara no buffer "Event"como um BM *Producer*.

O arquivo de opções usado por este programa é MinervaNetRecv.opts localizado no pacote DaqRecv package. Algumas linhas importantes são mostradas abaixo:

```
MEPManager.Buffers                  = { "Event" };
MEPManager.PartitionID              = 333;
MEPManager.PartitionBuffers         = false;
```

Estas linhas especificam a qual buffer nosso *producer* deve se conectar. O identificador de partição é 333, como de costume, e o nome do buffer não contém o identificador da partição .

O programa também tem as seguintes linhas:

```
ApplicationMgr.ExtSvc              += { "LHCb::DimDataReceiver/Receiver" };
ApplicationMgr.SvcOptMapping       += { "LHCb::DimDataReceiver/Receiver" };
```

```
Receiver.Buffer                    = "Event";
Receiver.UseEventRequests          = false;
Receiver.DeclareAsynchonously      = false;
```

O DimDataReceiver especificado aqui é o serviço GAUDI encarregado da recepção de *RawEvents* e da declaração deles no buffer "Event"buffer. Observe-se que não há qualquer especificação do *produtor de Rede (Network Producer)* . É o produtor de rede (NetSend) que se conecta a este consumidor de rede (NetRecv). Isto significa que para que a comunicação funcione, é preciso que o consumidor de rede seja lançado primeiro e somente então seja lançado o produtor de rede.

Reformatter

Este processo é uma tarefa *consumer* de BM. Ele se conecta ao buffer de eventos com o nome Reformatter. É lançado pelo *script* daq_recv.sh da seguinte maneira:

```
export UTGID=Reformatter;
${GAUDIONLINEROOT}/${CMTCONFIG}/Gaudi.exe
${GAUDIONLINEROOT}/${CMTCONFIG}/libGaudiOnline
OnlineTask
-auto
-main=${GAUDIONLINEROOT}/options/Main.opts
-opt=${MINERVAGAUDIONLINEROOT}/options/Reformatter.opts
-msgsvc=MessageSvc
```

É um programa GAUDI *online* com UTGID=Reformatter. O arquivo de opções usado por este programa contém:

```
ApplicationMgr.DLLs                += { "Gaucho" };
ApplicationMgr.ExtSvc              += { "MonitorSvc/MonitorSvc" };

ApplicationMgr.TopAlg               = { "Minerva::DecodeRawEvent/Reformatter" };
ApplicationMgr.TopAlg              += { "BasicHistos" };
```

Estas linhas especificam os algorítmos que rodarão para o processamento quase-*online* . Em janeiro de 2009 o MINERνA utiliza os algorítmos DecodeRawEvent e BasicHistos. O primeiro cria *RawDigits* a partir de *RawEvents*. O outro faz alguns histogramas a partir de *RawDigits*. Como temos o serviço MonitorSvc do GAUCHO carregado, todos estes histogramas

serão publicados como serviços DIM (ver seção C.2.3). Para adicionar algorítmos basta adicionar a seguinte linha após Basichistos:

```
ApplicationMgr.TopAlg                += { "MyAlgorithm" };
```

MyAlgorithm deve ler *RawEvents* ou *RawDigits* para fazer histogramas ou para processá-los em outra forma de objetos(*SupDigits*, *IDDigits*, *ODDigits*, etc.). Esta não é a maneira como o LHCb realiza seu processamento de dados. Eles não mudam o objeto *RawEvent*. Eles apenas mudam seu conteúdo (*RawBanks*).

As linhas restantes são muito similares as do arquivo de opções MinervaDiskWR.opts. As diferenças são : o Reformatter connecta-se e lê *RawEvents* do buffer "Event"e este buffer não contém o identificador de partição em seu nome. Tudo isto é especificado nas seguintes linhas.

```
MEPManager.Buffers                    = { "Event" };
MEPManager.PartitionBuffers           = false;
EventSelector.Input                   = "Event";
```

O Refformatter, da mesma forma que o DiskWR,também escreve objetos em disco. Estes objetos e o nome dos arquivos especificados:

```
ApplicationMgr.OutStream              = { "RawDigit" };
RawDigit.ItemList                     = { "/Event#1"
                                        , "/Event/DAQ#1"
                                        , "/Event/DAQ/Header#1"
                                        , "/Event/DAQ/RawDigits#1"};

OnlineEnv.RawDigitsFile                =
"DATAFILE='PFN:UnpackedData.root' TYP='POOL_ROOTTREE' OPT='RECREATE'";
OnlineEnv.RawHistosFile                = "basic_histos.root";

#include "/data_disk/rawdata/Output.opts"
RawDigit.Output                        = @OnlineEnv.RawDigitsFile;

// write histograms into a file
ApplicationMgr.HistogramPersistency    = "ROOT";
HistogramPersistencySvc.Outputfile     = @OnlineEnv.RawHistosFile;
```

Se tivermos que parar de enviar eventos para o quase-*online* nós, simplesmente, matamos o programa NetSend (simplesmente fechando a janela ou usando DIM para enviar o comando "unload"). Tudo o mais no nó continuará funcionando.

C.3.4 Controle e Monitoração de Processos

Controle de Processo

MINERνA utiliza o *script* runminervadaq.bat na máquina *online* para iniciar todos os programas *online* caso eles ainda não estejam rodando.

```
runminervadaq.bat \#events run\_number file_name
```

Os nomes dos parâmetros são auto explicativos. Detalhes sobre todos os parâmetros que este *script* pode aceitar encontram-se dentro do próprio *script*. Este *script* está localizado em

```
SITEROOT/minerva/MinervaScripts
```

Programas quase-*online* são iniciados pelo *script* daq_recv.sh na máquina quaseonl. Programas quase-*online* são monitorados pelo sistema de operação e, então, estão sempre rodando. Caso um programa pare de rodar, ele é reiniciado em cerca de um minuto. Para reiniciarmos todos os programas manualmente é preciso fazer o seguinte a partir da máquina quase-*online* :

```
ssh nearonline@minervadaq02.fnal.gov
/sbin/service daq_recv stop
/sbin/service daq_recv start
```

Se tudo é reiniciado em quase-*online* , então é preciso também interromper o programa NetSend em *online* de forma que o *script* runminervadaq.bat possa reiniciá-lo na tomada de dados seguinte. Isto se deve ao fato de que NetSend precisa reconectar-se ao NetRecv para a próxima transferência de evento.

O *script* daq_recv.sh e os *scripts* que mantêm os programas quase-*online* rodando automaticamente são encontrados em

```
DAQRECVROOT/scripts/
/home/nearonline/cmtuser/NearOnline_v5r1/DaqRecv/v5r1/scripts
```

O experimento MINERνA emprega o *script* runminervadaq.bat também para controlar programas *online* e quase-*online* . runminervadaq.bat não lida diretamente com transições de estado FSM, ele chama um segundo *script*, chamado fsm_status.bat, para realizar esta tarefa.

O *script* fsm_status.bat também é capaz de mudar o estado de qualquer programa *online* ou quase-*online* (exceto PROD_0) para o estado RUNNING. Não importa qual seja o estado corrente do programa, fsm_status vai mudá-lo para RUNNING. Caso o estado corrente seja RUNNING, o programa sera interrompido (estado NOT_READY) e, então, reiniciado novamente (estado RUNNING). Pode ser usado com as linhas de comando:

```
runminervadaq 0
fsm_status Reformatter
```

A primeira linha é apenas para colocar o ambiente em seu estado de trabalho. Após a execução da segunda linha, o estado atual do Reformatter será mostrado na linha de comando e será alterado de acordo com a figura C.3.

O *script* aceita dois parâmetros:stop e start.

```
runminervadaq 0
fsm_status Reformatter stop
```

A segunda linha altera o estado corrente do Reformatter para NOT_READY.

```
runminervadaq 0
fsm_status Reformatter start
```

A segunda linha espera até que o estado corrente do Reformatter seja RUNNING.

O *script* fsm_status.bat usa dois programas do pacote MinervaGaudiOnline, DimSendCommand.exe e DimQueryService.exe, para realizar sua função. Os códigos fontes podem ser encontrados em

```
${MINERVAGAUDIONLINEROOT}/main
```

Eles são escritos em C++ e empregam as classes definidas no pacote DIM. Eles são muito parecidos com os aplicativos encontrados no pacote DIM (ver seção C.2.1).

DimSendCommand.exe Permite enviar comandos para qualquer processo rodando *online* ou quase-*online* . Pode ser usado da seguinte maneira:

```
DimSendCommand.exe UTGID command
```

DimQueryService.exe Permite receber os dados publicados em um serviço DIM. Pode ser usado da seguinte maneira:

```
DimQueryService.exe UTGID/status
```

```
DimQueryService.exe UTGID/fsm_status
```

MINERνA emprega estes ao invés dos existentes no pacote DIM devido a uma possível incompatibilidade entre o DIM e o sistema WINDOWS. Isto, entretanto, ainda prescisa ser verificado. Uma vez que todos os programas (exceto PROD_0) são programas GAUDI *online* , eles podem receber comandos e mudar seus estados. Por exemplo, para enviar o comando "unload"ao programa Reformatter, o seguinte pode ser feito na máquina *online* :

```
runminervadaq 0
DimSendCommand.exe Reformatter unload
```

Isto mata o programa Reformatter e, então, após aproximadamente um minuto, todos os programas no quase-*online* reiniciarão.

O *script* fsm_status.bat lê os serviços UTGID/status e UTGID/fsm_status várias vêzes de forma a realizar a transição na ordem correta como visto na figura C.3. Se qualquer transição falha mais de 10 vêzes, todos os programa *online* e quase-*online* serão reiniciados. Para mais detalhes, o *script* pode ser encontrado em

```
SITEROOT/minerva/MinervaScripts
```

O *script* fsm_status.bat deve ser substituído por um aplicativo C++ mais robusto e mais rápido. Com isto seremos capazes de usar todas as vantagens oferecidas pelo DIM e os aplicativos DimSendCommand.exe e DimQueryService.exe não mais serão necessários.

Monitoração de processos

Podemos empregar a ferramenta DIM mencionada em C.2.1 para o monitoramento remoto de processos *online* e quase-*online*. Pode ser iniciado de qualquer máquina que "veja"a máquina quase-*online* (pois esta é a máquina que roda o DNS). Em uma máquina com sistema linux e que tenha o *software* do MINERνA podemops fazer o seguinte:

```
source ${SITEROOT}/lhcb/ONLINE/ONLINE_v4r3/Online/DIM/v16r13/cmt/setup.sh
export DIM_DNS_NODE=minervadaq02.fnal.gov
${DIMROOT}/${CMTCONFIG}/did.exe &
```

A primeira linha prepara o ambiente para DIM. A segunda define onde o DNS roda e a terceira lanca o aplicativo. Após isto, a janela vista na figura C.11 aparecerá após escolhermos "All Servers"no menu.

DID mostrará todos os servidores (*jobs* GAUDI *online*), pelos seus UTGID, conectados ao DNS e permitirá a leitura de qualquer serviço neles alé de permitir o envio de comandos (ver

D.1 Usuário NearOnline

Para executar o que é descrito nesta seção é necessário trabalhar em uma máquina que possa "ver"minervadaq02.fnal.com (por exemplo minerva01) e ter acesso a uma distribuição do *software* do MINERνA que contem o projeto ONLINE do LHC. No momento, a única distribuição que satisfaz estas especificações e distribuição v5r1 instalada no minervadaq02.

```
/data_disk/release/v5r1
```

Antes de seguir os passos aqui descritos é necessário fazer o seguinte uma vez em cada seção :

- Estabelecer o ambiente para o *software* do MINERνA .

```
source /data_disk/release/v5r1/setup.sh v5r1 /data_disk/release/v5r1
```

- Estabelecer o ambiente para o projeto NearOnline.

```
cd ~/cmtuser
setenvNearOnline v5r1
```

- Pegar o pacote DaqRecv.

```
getpack DaqRecv v5r1
```

D.1.1 Monitorando Histogramas

O pacote DaqRecv contém um *script* para monitoramento de histogramas empregando o cliente HPDMonitor do projeto ONLINE do LHCb. para usá-lo é preciso:

- Estabelecer o ambiente para o pacote DaqRecv.

```
source ${DAQRECVROOT}/cmt/setup.sh
```

- Source the Monitor.sh script.

```
source ${DAQRECVROOT}/scripts/Monitor.sh
```

Uma vez lançado, aparecerá um GUI will (ver figura D.2). Acione o botão "connect", selecione, na árvore à direita, os histogramas que você quer monitorar, escolha algumas das opções de histogramas no quadro de "setup"e acione o botão "select". Os histogramas com as opções escolhidas aparecerão na grande janela. Os histogramas começarão a serem atualizados, em tempo real, no momento em que o botão "start"seja acionado.

D.2.1 Publicando histogramas

Primeiro é necessário criar um algorítmo GAUDI que crie histogramas em sua área. Use o pacote Histogramatron dentro do projeto ENDEAVOR:

- Estabeleça o ambiente para o projeto ENDEAVOR.

```
cd ~/cmtuser
setenvEndeavor v5r1
```

- Estabeleça o ambiente para o pacote Histogramatron.

```
getpack Histogramatron v5r1
cd Histogramatron/v5r1
source cmt/setup.sh
```

- Crie seu código e tente compilá-lo. Exemplos podem ser encontrados dentro do pacote.

```
cd src
emacs MyAlgorithm.cpp, Myalgorithm.h
cd ../options
emacs MyAlgorithm.opts
cd ../cmt
make
```

- Teste seu algorítmo

```
cd ..
${GAUDIROOT}/${CMTCONFIG}/Gaudi.exe options/MyAlgorithm.opts
```

Se tudo correr bem, você pode adicionar seu algorítmo ao processamento de dados quase-*online* :

- Carregue seu código no CVS.

```
cd $HISTOGRAMATRON/src
cvs add MyAlgorithm.cpp MyAlgorithm.h
cvs commit
```

- Log to minervadaq02 as nearonline and update the Histogramatron package.

```
ssh nearonline@minervadaq02.fnal.gov
cd cmtuser/Endeavor_v5r1/Histogramatron
cvs update
```

- Open the "Reformatter.opts"options file

```
cd ~/cmtuser/NearOnline_v5r1/DaqRecv/v5r1/options
emacs Reformatter.opts
```

- And add the following line to it

```
ApplicationMgr.TopAlgorithms += {MyAlgorithm};
```

Após isto, seu algorítmo estará rodando quase-*online* e os histogramas resultantes serão publicados e armazenados como arquivo ROOT em

```
/data_disk/data/*_RawHistos.root
```

D.2.2 Controle de Processos

Todos os processos podem ser iniciados ou parados com os seguintes comandos:

```
/sbin/service daq_recv start
/sbin/service daq_recv stop
```

Processos quase-*online* são monitorados a cada minuto pelo sistema para verificar se estão rodando [1]. Se algum deles para ou morre, o sistema reiniciará TODOS eles. Para desativar esta propriedade, é preciso remover o arquivo crontab corrente da seguinte maneira:

```
crontab -r
```

Para reativar esta propriedade:

```
crontab /home/nearonline/crontab.file
```

Para verificar o conteúdo corrente do arquivo crontab, basta usar o comando:

```
crontab -l
```

Quando a característica está habilitada obtemos o seguinte resultado:

```
# every minute, see if the daq_recv process is running and if not restart it
/home/nearonline/NearOnline_v5r1/DaqRecv/v5r1/scripts/keep_running_daq_recv.sh
```

Por outro lado, quando a característica está desabilitada obtemos:

```
no crontab for nearonline
```

[1] Este monitor é um programa cron em minervadaq02

D.2.3 Arquivos de registro de atividades

Processos quase-*online* (exceto o DNS.exe) registram todas as atividades em um arquivo localizado em:

```
/home/nearonline/log/{Name}.log
```

con o nome

```
{Name}.log
```

onde "Name"é o nome do processo quase-*online* . Pode ser "Reformatter", "InputBuffer"ou "NetRecv".

Assim que os processos quase-*online* param, todos os arquivos log são renomeados da seguinte maneira:

```
{Name}_{year}_{month}_{day}_{hour}_{minute}_{seconds}.log
```

D.2.4 Atualizações

Se, por alguma razão , a localização de onde se encontram as distribuições do *software* do MINERνA ou os pacotes do DaqRecv, é necessário editar os seguintes arquivos:

- Atualizar a localização onde se encontra a pasta com os *scripts* do DaqRecv na seguinte linha:

  ```
  dir=/home/nearonline/NearOnline_v5r1/DaqRecv/v5r1/scripts
  ```

  do arquivo:

  ```
  /etc/init.d/daq_recv
  ```

- Atualizar a localização onde se encontra a pasta do *script* na seguinte linha:

  ```
  /home/nearonline/NearOnline_v5r1/DaqRecv/v5r1/scripts/keep_running_daq_recv.sh
  ```

  do arquivo:

  ```
  /home/nearonline/crontab.file
  ```

- Atualizar a versão do *software* do MINERνA e a localização de onde ele esta instalado alterando as seguintes linhas

```
export RELEASEVERSION=v5r1
export RELEASEAREA=/data_disk/release/v5r1
```

do arquivo:

```
/home/nearonline/cmtuser/setup.sh
```

D.2.5 Notas adicionais

Há alguns problemas que podem ser facilmente resolvidos em distribuições futuras:
Ao fazer

```
cd NearOnline_v5r1/DaqRec/v5r1
source cmt/setup.sh
```

Obtemos o seguinte resultado:

```
CMT>Project LHCB requested with conflicting releases LHCB_v23r1 and LHCB_v22r10
CMT>Warning: package Histogramatron v*  not found (requested by DaqRecv)
```

Isto é "normal"e todos os *scripts* do DaqRecv funcionarão . O primeiro alerta deve-se a uma incompatibilidade na versão do projeto, como pode ser visto na tabela D.1.

Projeto	depende de	depende de	depende de
NEARONLINE_arturo_2009_01_25	ENDEAVOR_v5r1	MINERVA_v5r1	LHCB_22r10
-	ONLINE_v4r3	LHCB_23r1	-

Tabela D.1: Dependência básica de projetos do projeto NearOnline

O segundo alerta ocorre porque o Histogramatron ainda não é parte da distribuição corrente do *software* do MINERνA (v5r1). Isto pode ser evitado da mesma forma feita "User release area"do nearonline's com os dois projetos (NearOnline e Endeavor).

Scripts adicionais são descritos brevemente a seguir:

daq_recv.sh Este *script* lança todos os processos quase-*online* enviando o resultado para um arquivo. Ele não deve ser usado. Este *script* é chamado pelo comando "/sbin/service daq_recv start"command.

daq_recv_test.sh Este *script* também lançará todos os processos quase-*online* mas seus resultados serão mostrados em uma janela xterm. Este *script* pode ser rodado uma vez desabilitado o programa cron.

keep_running_daq_recv.sh Este é o *script* que é chamado pelo programa cron a cada minuto.

monitor_daq_recv.sh Este *script* é usado pelo *script* keep_running_daq_recv.sh para ver se um processo não está rodando.

kill_daq_recv.sh Este é o *script* que é chamado pelo comando "/sbin/service daq_recv stop".

daq_recv_opts.sh Algumas definições comuns para os *scripts* anteriores.

Apêndice E

Códigos fontes desenvolvidos

E.1 O algorítmo Montest

E.1.1 MonTestAlg.h

```
#ifndef MINERVA_MONTESTALG_H
#define MINERVA_MONTESTALG_H
#include "GaudiAlg/GaudiHistoAlg.h"


namespace Minerva
{
        class MonTestAlg : public GaudiHistoAlg
        {
        public:
                MonTestAlg(const std::string& name, ISvcLocator* pSvcLocator);
                virtual StatusCode initialize();
                virtual StatusCode execute();
                virtual StatusCode finalize();
                virtual ~MonTestAlg() { };
        private:
                int m_counter;
        };


}
#endif
```

E.1.2 MonTestAlg.cpp

```
#include "MonTestAlg.h"
#include "Event/RawDigit.h"

namespace Minerva {

	MonTestAlg::MonTestAlg(const std::string& name, ISvcLocator* pSvcLocator
	    )
		: GaudiHistoAlg(name, pSvcLocator){
		m_counter = 0;
	}

	StatusCode MonTestAlg::initialize() {
		info() << "MonTestAlg::initialize..." << endreq;
		declareInfo("counter",m_counter,"Counter value");
		return StatusCode::SUCCESS;
	}

	StatusCode MonTestAlg::execute(){

		info() << "MonTestAlg::execute..." << endreq;
		Minerva::RawDigits* digits = get<Minerva::RawDigits>( evtSvc(),
		    Minerva::RawDigitLocation::Default);
		for (Minerva::RawDigits::iterator it = digits->begin(); it !=
		    digits->end(); ++it) {
			m_counter++;
			plot1D((*it)->qhi(),"charge",0,3500,70);
		}
		return StatusCode::SUCCESS;
	}

	StatusCode MonTestAlg::finalize(){

		info() << "MonTestAlg::finalize..." << endreq;
		info() << "counter = " << m_counter << endreq;
		return StatusCode::SUCCESS;
	}
}
#include "GaudiKernel/DeclareFactoryEntries.h"
DECLARE_NAMESPACE_ALGORITHM_FACTORY(Minerva, MonTestAlg)
```

E.2 O algorítmo Publisher

E.2.1 PublisherAlg.h

```
#ifndef PUBLISHERALG_H
#define PUBLISHERALG_H

#include <string>
#include "Event/RawDigit.h"
#include "Event/DAQHeader.h"
#include "GaudiAlg/GaudiAlgorithm.h"

class PublisherAlg : public GaudiAlgorithm {

public:
  PublisherAlg(const std::string& name, ISvcLocator* pSvcLocator);
  virtual StatusCode initialize();
  virtual StatusCode execute();
  virtual StatusCode finalize();
  virtual ~PublisherAlg() {};

private:
  // the size of the array that contains one event
  // 10 PMTs and 64 pixel/PMT
  // (640 digits + 1 header per event) size = 14*640+13 = 8973 integers
  int m_size;

  // the format of the array that contains one event
  std::string m_format;

  // the array that contains one event
  int* m_data;

  DimService* m_eventservice;

  }; // end of class

// helper function: converts DAQHeader and RawDigits into an array of integers
int EventtoArray( Minerva::DAQHeader *head, Minerva::RawDigits *digits, int*
    data );

#endif
```

E.2.2 PublisherAlg.cpp

```
#include "dis.hxx"
#include "PublisherAlg.h"

#include "GaudiKernel/DeclareFactoryEntries.h"
#include "GaudiKernel/AlgFactory.h"
DECLARE_ALGORITHM_FACTORY( PublisherAlg );

PublisherAlg::PublisherAlg(const std::string& name, ISvcLocator* pSvcLocator)
  : GaudiAlgorithm(name, pSvcLocator)
  , m_size(8973)
  , m_format("I")
  , m_data(0) {
}

StatusCode PublisherAlg::initialize() {

  StatusCode sc = GaudiAlgorithm::initialize(); // must be executed first
  if ( sc.isFailure() ) return sc;  // error printed already by GaudiAlgorithm
  info() << "PublisherAlg::initialize..." << endreq;

  // the array we want to send
  // we can not change this pointer!!!
  m_data = new int[m_size];

  // declare the service that will send data
  //declareInfo("eventcounter",m_eventcounter,"eventcounter");
  //declareInfo("event", m_format, m_data, m_size*sizeof(int), "event");
  // to do: create a class that inherits from DimService and calculate size
  m_eventservice = new DimService("TestServer/Publisher/events", "I", m_data,
      m_size*sizeof(int));

  return StatusCode::SUCCESS;
}

StatusCode PublisherAlg::execute() {

  // Get DAQHeader from the TES
  Minerva::DAQHeader*
  head = get<Minerva::DAQHeader>(Minerva::DAQHeaderLocation::Default);

  // Get RawDigits from the TES
  Minerva::RawDigits*
```

```
  digits = get<Minerva::RawDigits>(Minerva::RawDigitLocation::Default);

  // Convert all event information into an array to be sent by DIM
  EventtoArray( head, digits, m_data );

  // Send the array
  m_eventservice -> updateService();

  // Show event information
  debug() << "number of event    " << m_data[3] << endreq;

  return StatusCode::SUCCESS;
}

StatusCode PublisherAlg::finalize() {

  info() << "PublisherAlg::finalize..." << endreq;

  return GaudiAlgorithm::finalize();  // must be called after all other actions
}

// helper function
// converts DAQHeader and RawDigits into an array of integers
// what should this function return??
int EventtoArray( Minerva::DAQHeader* head, Minerva::RawDigits* digits, int*
    data ) {

  // Read event information from DAQHeader
  data[0]  = head -> runNumber();
  data[1]  = head -> eventType();
  data[2]  = head -> spillNumber();
  data[3]  = head -> eventNumber();
  data[4]  = head -> gpsTime();
  data[5]  = head -> beamType();
  data[6]  = head -> beamEnergy();
  data[7]  = head -> beamIntensity();
  data[8]  = head -> calibrationType();
  data[9]  = head -> calibrationSetting();
  data[10] = head -> triggerInfo();
  data[11] = head -> readoutErrors();
  data[12] = head -> readoutType();

  // Read event information from each rawdigit in RawDigits (ndigits rawdigits
     readed)
```

```
  int ndigits = 0;
  for ( Minerva::RawDigits::iterator it = digits->begin();
        it != digits->end();
        ++it ) {

    // The iterator is a pointer to rawdigit*
    Minerva::RawDigit* rawdigit = *it;

    data[13 + 14*ndigits] = rawdigit -> channelID().link();
    data[14 + 14*ndigits] = rawdigit -> channelID().crate();
    data[15 + 14*ndigits] = rawdigit -> channelID().croc();
    data[16 + 14*ndigits] = rawdigit -> channelID().chain();
    data[17 + 14*ndigits] = rawdigit -> channelID().board();
    data[18 + 14*ndigits] = rawdigit -> channelID().pixel();
    data[19 + 14*ndigits] = rawdigit -> channelID().hit();
    data[20 + 14*ndigits] = rawdigit -> discrFired();
    data[21 + 14*ndigits] = rawdigit -> tdc();
    data[22 + 14*ndigits] = rawdigit -> ticks();
    data[23 + 14*ndigits] = rawdigit -> systicks();
    data[24 + 14*ndigits] = rawdigit -> qlo();
    data[25 + 14*ndigits] = rawdigit -> qmed();
    data[26 + 14*ndigits] = rawdigit -> qhi();

    // one rawdigit has been read
    ++ndigits;

  } // end for

  return 0;

}
```

E.3 O algorítmo Receiver

E.3.1 ReceiverAlg.h

```
#ifndef RECEIVERALG_H
#define RECEIVERALG_H

#include "dic.hxx"
#include "Event/RawDigit.h"
#include "Event/DAQHeader.h"

#include "DimInfoCounter.h"
#include "DimInfoEvent.h"

#include "GaudiAlg/GaudiAlgorithm.h"

class ReceiverAlg : public GaudiAlgorithm {

public:
  ReceiverAlg(const std::string& name, ISvcLocator* pSvcLocator);
  virtual StatusCode initialize();
  virtual StatusCode execute();
  virtual StatusCode finalize();
  virtual ~ReceiverAlg() {};

private:
  int m_counter;
  DimInfoEvent* m_event;
  DimInfoCounter* m_eventcounter;
  std::string m_rawDigitLocation;

};

#endif
```

E.3.2 ReceiverAlg.cpp

```
#include "ReceiverAlg.h"

#include "GaudiKernel/DeclareFactoryEntries.h"
#include "GaudiKernel/AlgFactory.h"
DECLARE_ALGORITHM_FACTORY( ReceiverAlg );

ReceiverAlg::ReceiverAlg(const std::string& name, ISvcLocator* pSvcLocator)
  : GaudiAlgorithm(name, pSvcLocator) {

  //declareProperty( "RawDigitLocation", m_rawDigitLocation = Minerva::
      RawDigitLocation::Default);
}

StatusCode ReceiverAlg::initialize() {

  StatusCode sc = GaudiAlgorithm::initialize(); // must be executed first
  if ( sc.isFailure() ) return sc;  // error printed already by GaudiHistoAlg
  info() << "ReceiverAlg::initialize..." << endreq;

  // suscribe to services
  m_event = new DimInfoEvent("TestServer/Publisher/events", 5, 1);
  //m_eventcounter = new DimInfoCounter("TestServer/Publisher/eventcounter", 1,
      1);

  return StatusCode::SUCCESS;
}

StatusCode ReceiverAlg::execute() {

  // wait until service is updated
  int i = 0;
  while( ! m_event -> serviceUpdated() ) {
    ++i;
  }

  //how much am I waiting?
  info() << " number of wait loops:  " << i << endreq;

  // get Minerva::DAQHeader and Minerva::RawDigits from DimInfoEvent
  Minerva::DAQHeader* head;
  Minerva::RawDigits* digits;
  m_event -> get( head, digits );
```

```
//    std::pair<Minerva::DAQHeader*, Minerva::RawDigits*>* event = m_event ->
    getEvent();
//    head   = event -> first;
//    digits = event -> second;
  //head   = m_event -> getHeader();
  //digits = m_event -> getRawDigits();

  // show information about the event received
  debug() << " header and rawdigits received!!!" << endreq;
  debug() << " event number received:    " << head    -> eventNumber()  <<
      endreq;
  debug() << " size of the event queue:  " << m_event -> m_queue.size() <<
      endreq;

  // put Minerva::DAQHeader and Minerva::RawDigits in the TES
  put(head, Minerva::DAQHeaderLocation::Default);
  put(digits, Minerva::RawDigitLocation::Default);

  // get ready to receive the next one
  //m_event -> ResetServiceUpdated();

  return StatusCode::SUCCESS;

}

StatusCode ReceiverAlg::finalize() {

  return GaudiAlgorithm::finalize();

}
```

E.4 A classe DimInfoEvent

E.4.1 DimInfoEvent.h

```
// $Id: DimInfoHisto.h,v 1.4 2007/05/31 15:21:49 ukerzel Exp $
#ifndef DIMINFOEVENT_H
#define DIMINFOEVENT_H 1

// Include files
#include <string>
#include <queue>

#include "dic.hxx"
#include "Event/RawDigit.h"
#include "Event/DAQHeader.h"

/** @class DimInfoEvent DimInfoEvent.h
 *
 *  @author Arturo Fiorentini
 *  based on DimInfoHistos.h by Ulrich Kerzel
 *  @date   2008-07-28
 *
 * Format returned by getData()
 * DAQHeader (1 per event)   RunNumber, EventType, SpillNumber, EventNumber, GpsTime, BeamType,
 *                           BeamEnergy, BeamIntensity, CalibrationType, CalibrationSetting,
 *                           TriggerInfo, ReadoutErrors, ReadoutErrors, ReadoutType
 *
 * RawDigits (several per event) link, crate, croc, chain, board, pixel, hit,, DiscrFired, Tdc
 *                               Ticks, Systicks, setQlo, Qmed, Qhi
 *
 */

class DimInfoEvent : public DimInfo{
public:

  DimInfoEvent( std::string serviceName
                , int refreshTime
                , int verbosity = 0);

  virtual ~DimInfoEvent( );
```

```
    Minerva::DAQHeader* getHeader();
    Minerva::RawDigits* getRawDigits();
    std::pair<Minerva::DAQHeader*, Minerva::RawDigits*>* DimInfoEvent::getEvent();

    bool serviceOK();
    bool serviceUpdated();
    void ResetServiceUpdated();
    std::queue< std::pair<Minerva::DAQHeader*, Minerva::RawDigits*>* > m_queue;

    void DimInfoEvent::get( Minerva::DAQHeader* &head, Minerva::RawDigits* &digits
        );

private:
    // Called whenever server updates information,  from DimInfo
    void        infoHandler();

    // Interprets data received
    void        setData();

    // Status of the service updated
    bool        m_serviceOK;

    // change to true once call-back is received and values may have changed
    bool        m_serviceUpdated;

    int         m_verbosity;
    int         m_serviceSize;
    std::string m_serviceName;
    int*        m_data;

    //Minerva::DAQHeader* m_head;
    //Minerva::RawDigits* m_digits;
    //std::pair<Minerva::DAQHeader*, Minerva::RawDigits*>* m_event;

    int         m_nWait;          // count how often waited for service.
    int         m_waitTime;       // wait for service in usec

};

#endif // DIMINFOEVENT_H
```

E.4.2 DimInfoEvent.cpp

```
// Include files
#include <iostream>
#include <stdlib.h>
#include <malloc.h>

#ifdef WIN32
namespace win {
#include <windows.h>
}
#define mysleep win::Sleep
#else
#define mysleep usleep
#endif

#include "DimInfoEvent.h"

//-----------------------------------------------------------------------------
// Implementation file for class : DimInfoEvent
//                                 based on DimInfoHisto
//
// 2008-07-28 : Arturo Fiorentini
//              based on code by Ulrich Kerzel
//-----------------------------------------------------------------------------

DimInfoEvent::DimInfoEvent( std::string serviceName
                          , int refreshTime
                          , int verbosity )
  : DimInfo(serviceName.c_str(), refreshTime, -1)
  , m_serviceOK(false)
  , m_serviceUpdated(false)
  , m_verbosity(verbosity)
  , m_serviceSize(0)
  , m_serviceName(serviceName)
  , m_nWait(0)
  , m_waitTime(0) {

  if (m_verbosity > 0)
    std::cout << "*** initialising DIM for service " << m_serviceName
              << std::endl;

  // check if service exists
  DimBrowser dimBrowser;
```

```
    int nServices = dimBrowser.getServices(m_serviceName.c_str());
    if (m_verbosity > 0)
      std::cout << "# services for this service name " << nServices
                << std::endl;

    if (nServices != 0) {

      // Server publishes the counter requested.
      m_serviceSize = getSize()/sizeof(int);

      // wait until data has arrived
      m_waitTime = (int)((float)refreshTime*1e5); // 10% of refresh time
      if (m_verbosity > 0)
        std::cout << "wait time " << m_waitTime << std::endl;

      while (m_serviceSize <= 0) {
        m_nWait++;
        std::cout << m_nWait << "th wait for service " << m_serviceName << std::
            endl;
        mysleep(m_waitTime);
        m_serviceSize = getSize()/sizeof(int);
      }

      m_serviceOK   = true;
      m_nWait       = 0; //reset

      // assume integer type by default, now check for other types
      char* dimService;
      char* dimFormat;
      int   dimType;
      while(dimType = dimBrowser.getNextService(dimService, dimFormat)) {
        if (m_verbosity > 0){
          std::cout << "DIM service : " << dimService << " format " << dimFormat
              << std::endl;
        } //verbosity
      } //while
    } //if nServices

     if (m_verbosity > 0) {
       std::cout << "service OK?       " << m_serviceOK      << std::endl;
       std::cout << "        size      " << m_serviceSize    << std::endl;
     } // if verbosity

  } //constructor
```

```
85
// Destructor
DimInfoEvent::~DimInfoEvent() {}

void DimInfoEvent::infoHandler() {
90
  // wait until data has arrived
  m_serviceSize = getSize()/sizeof(int);
  while (m_serviceSize <= 0) {
    m_serviceOK = false;
95  m_nWait++;
    std::cout << m_nWait << "th wait for service " << m_serviceName << std::endl
        ;
    mysleep(m_waitTime);
    m_serviceSize = getSize()/sizeof(int);
  } //while
100
  m_serviceOK = true;
  m_nWait     = 0;

  // Receive information from DIM
105 m_data      = (int*) getData();

  DimInfoEvent::setData();

  if (m_verbosity > 0)
110   std::cout << "DimInfoEvent::infoHandler service has been updated " << std::
        endl;
  m_serviceUpdated = true;

}

115 void DimInfoEvent::setData() {

  // create a Minerva::DAQHeader
  // call default contructor
  Minerva::DAQHeader* head = new Minerva::DAQHeader();
120
  // fill the Minerva::DAQHeader with the information received
  head -> setRunNumber(          m_data[0]  );
  head -> setEventType(          m_data[1]  );
  head -> setSpillNumber(        m_data[2]  );
125 head -> setEventNumber(        m_data[3]  );
  head -> setGpsTime(            m_data[4]  );
```

```
    head -> setBeamType(            Minerva::DAQHeader::BeamTypes(m_data[5])   );
    head -> setBeamEnergy(          Minerva::DAQHeader::BeamEnergies(m_data[6])   );
    head -> setBeamIntensity(       m_data[7]   );
    head -> setCalibrationType(     Minerva::DAQHeader::CalibrationTypes(m_data[8])
        );
    head -> setCalibrationSetting( m_data[9]   );
    head -> setTriggerInfo(         m_data[10] );
    head -> setReadoutErrors(       Minerva::DAQHeader::ReadoutErrors(m_data[11]) )
       ;
    head -> setReadoutType(         Minerva::DAQHeader::ReadoutTypes(m_data[12]) );

    // create a Minerva::RawDigits
    // call default contructor
    Minerva::RawDigits* digits = new Minerva::RawDigits();

    // read the information received from DIM (ndigits rawdigits created)
    for( int ndigits = 0; ndigits != 640 ; ++ndigits ) {

      // create a channelID
      int link  = m_data[13 + 14*ndigits];
      int crate = m_data[14 + 14*ndigits];
      int croc  = m_data[15 + 14*ndigits];
      int chain = m_data[16 + 14*ndigits];
      int board = m_data[17 + 14*ndigits];
      int pixel = m_data[18 + 14*ndigits];
      int hit   = m_data[19 + 14*ndigits];
      Minerva::ChannelID chID( link, crate, croc, chain, board, pixel, hit );

      // create one digit
      Minerva::RawDigit* digit = new Minerva::RawDigit();
      digit -> setDiscrFired( m_data[20 + 14*ndigits] );
      digit -> setTdc(        m_data[21 + 14*ndigits] );
      digit -> setTicks(      m_data[22 + 14*ndigits] );
      digit -> setSysticks(   m_data[23 + 14*ndigits] );
      digit -> setQlo(        m_data[24 + 14*ndigits] );
      digit -> setQmed(       m_data[25 + 14*ndigits] );
      digit -> setQhi(        m_data[26 + 14*ndigits] );

      // fill the Minerva::RawDigits
      digits->insert( digit, chID );

    } // end for

    // put header and digits into a std::pair
```

```
    std::pair<Minerva::DAQHeader*, Minerva::RawDigits*>* event = new std::pair<
        Minerva::DAQHeader*, Minerva::RawDigits*>( head, digits );

    // put the std::pair in the queue
    m_queue.push ( event );

}

void DimInfoEvent::get( Minerva::DAQHeader* &head, Minerva::RawDigits* &digits )
    {

    while ( m_queue.empty() ) {
      // Wait for data
      mysleep(m_waitTime);
    }

    //get head and digits
    head = m_queue.front() -> first;
    digits = m_queue.front() -> second;

    // delete the last event
    m_queue.pop();

}

std::pair<Minerva::DAQHeader*, Minerva::RawDigits*>* DimInfoEvent::getEvent() {

    // wait until data is ready
    while ( m_queue.empty() ) {}

    // return the next event in the queue
    return m_queue.front();

}

Minerva::DAQHeader* DimInfoEvent::getHeader() {

    return m_queue.front() -> first;

}

Minerva::RawDigits* DimInfoEvent::getRawDigits() {

```

```
    //mysleep(m_waitTime);

    return m_queue.front() -> second;

}

bool DimInfoEvent::serviceOK() {
  return m_serviceOK;
}

bool DimInfoEvent::serviceUpdated() {
  return m_serviceUpdated;
}

void DimInfoEvent::ResetServiceUpdated() {

  // delete the last event
  m_queue.pop();
  m_serviceUpdated = false;

  if (m_verbosity > 0)
    std::cout << "DimInfoEvent set flag serviceUpdated to " << m_serviceUpdated
        << std::endl;

}
```

E.5 O *script* usado para iniciar os processos no sistema *online*

E.5.1 runminervadaq.bat

```
@echo off

rem This version of the minerva daq script expects at least three arguments (if the first is not equal to zero).
rem  arg1==#events; arg2==run# (pseudo-dummy); arg3==base file name for log and data (contains all "real" run #'s)
rem The script can additionally accept a fourth argument arg4==Hardware Init Level (0==full==default, 1==minimal)
rem The script can additionally accept a fifth argument arg5==Operating Mode (0==OneShot(ped/LI)==default, 1==cosmic
rem The script can now also accept a sixth argument arg6==Max Run Time in seconds (0 == infinity)
rem Because arg1 actually specifies the maximum number of events, the DAQ stops readout according to the first condition met.

:start

@echo Setting Environment...
@echo Parameters to minervadaq: %*
@if not defined CMTSITE call %MYSITEROOT%\minerva\MinervaScripts\ExtCMT.bat
@pushd %MYSITEROOT%\minerva\MINERVAONLINE\MINERVAONLINE_v2r0\MinervaOnline\MinervaDAQ\v4r0\cmt
@if not defined MINERVADAQROOT call setup.bat
@pushd %MYSITEROOT%\minerva\MINERVAONLINE\MINERVAONLINE_v2r0\MinervaOnline\MinervaGaudiOnline\v3r0\cmt
@if not defined MINERVAGAUDIONLINEROOT call setup.bat
@if not defined TAN_PORT call %MINERVAGAUDIONLINEROOT%\scripts\preamble.bat
@set OPTS=%GAUDIONLINEROOT%\options
@set DIM_DNS_NODE=192.168.1.1
popd
popd
rem If no events requested, exit the script
if %1==0 (
  EXIT /B
)

echo Creating directory for data...
rem pushd c:\home\data
```

```
pushd z:\rawdata
echo OnlineEnv.RawEventsFile = "DATAFILE='PFN:%3_RawEvents.inprogress' TYP='
    POOL_ROOTTREE' OPT='RECREATE'"; >  Output.opts
echo OnlineEnv.RawDigitsFile = "DATAFILE='PFN:%3_RawDigits.inprogress' TYP='
    POOL_ROOTTREE' OPT='RECREATE'"; >> Output.opts
echo OnlineEnv.RawHistosFile = "%3_RawHistos.inprogress"; >> Output.opts
echo OnlineEnv.DSTFile       = {"FILE1 DATAFILE='%3_DST.root' TYP='ROOT' OPT='
    NEW'"}; >> Output.opts

echo Starting processes...

rem Check whether MBMMON is running
set _=
set UTGID=MBMMon
for /F "usebackq delims=" %%i in (`tasklist /FI "WINDOWTITLE eq %UTGID%" /FO "
    CSV" /NH ^| find "Gaudi.exe" /C`) do set _=%%i
if %_%==0 (
   echo Starting %UTGID%...
   start "%UTGID%" %GAUDIONLINEROOT%\%CMTCONFIG%\Gaudi.exe OnlineKernel.dll
       mbm_mon
) else (
   echo %UTGID% is already running...
)
set UTGID=

set _=
set UTGID=OutputBuffer
for /F "usebackq delims=" %%i in (`tasklist /FI "WINDOWTITLE eq %UTGID%" /FO "
    CSV" /NH ^| find "Gaudi.exe" /C`) do set _=%%i
if %_%==0 (
   echo Starting %UTGID%...
   start "%UTGID%" %gaudi_exe% -main=%OPTS%\MBMinit.opts -opt=%OPTS%\Daemon.opts
       -msgsvc=%msg_svc%
   sleep 10
   call fsm_status %UTGID% start
) else (
   echo %UTGID% is already running...
)
set UTGID=

set UTGID=Reformatter
   rem assume this process is always running in minervadaq02
   echo %UTGID% is already running...
   call fsm_status %UTGID%
```

```
if [!error!]==true goto killgaudi
set UTGID=

set UTGID=NetRecv
   rem assume this process is always running in minervadaq02
   echo %UTGID% is already running...
   rem call fsm_status %UTGID%
set UTGID=

set _=
set UTGID=NetSend
for /F "usebackq delims=" %%i in (`tasklist /FI "WINDOWTITLE eq %UTGID%" /FO "
   CSV" /NH ^| find "Gaudi.exe" /C`) do set _=%%i
if %_%==0 (
   echo Starting %UTGID%...
   start "%UTGID%" %gaudi_exe% -main=%OPTS%\Main.opts -opt=%
      MINERVAGAUDIONLINEROOT%\options\MinervaNetSend.opts -msgsvc=%msg_svc%
   sleep 10
   echo %UTGID% started... checking FSM status
   call fsm_status %UTGID% start
) else (
   echo %UTGID% is already running... checking FSM status
   rem call fsm_status %UTGID%
)
set UTGID=
if [!error!]==true goto killallgaudi

set _=
set UTGID=DiskWR
for /F "usebackq delims=" %%i in (`tasklist /FI "WINDOWTITLE eq %UTGID%" /FO "
   CSV" /NH ^| find "Gaudi.exe" /C`) do set _=%%i
if %_%==0 (
   echo Starting %UTGID%...
   start "%UTGID%" %gaudi_exe% -main=%OPTS%\Main.opts -opt=%
      MINERVAGAUDIONLINEROOT%\options\MinervaDiskWR.opts -msgsvc=%msg_svc%
   sleep 10
   echo %UTGID% started... checking FSM status
   call fsm_status %UTGID% start
) else (
   echo %UTGID% is already running... checking FSM status
   call fsm_status %UTGID%
)
set UTGID=
if [!error!]==true goto killallgaudi
```

```
echo Launching DAQ...
rem sleep 10
start /REALTIME /W MinervaDaqMain.exe %*

echo Stopping processes...
call fsm_status DiskWR stop
call fsm_status Reformatter stop
rem call fsm_status NetRecv stop
goto end

rem Kill all open Gaudi processes
:killallgaudi
taskkill /FI "WINDOWTITLE eq NetSend"
taskkill /FI "WINDOWTITLE eq OutputBuffer"
taskkill /FI "WINDOWTITLE eq MBMMon"
popd
pushd c:\home\data\
echo DiskWR Freeze > Crash%3.log
popd
rem 1 minute for the processes in minervadaq02 to restart
sleep 60

goto start

rem End of script.
:end
set UTGID=
set _=
set DISKWR_STATE=
pushd z:\data
rem The sleep period may need to be extended to allow the renaming process to
    work (root file must be free to be renamed).
rem sleep 5
rename %3_RawDigits.inprogress %3_RawDigits.root
rename %3_RawHistos.inprogress %3_RawHistos.root
popd
rename %3_RawEvents.inprogress %3_RawEvents.root
```

E.5.2 fsm_status.bat

```
@echo off
rem the first parameter is the name of the online process we want to deal with
set UTGID=%1
set flag=%2
set STOPPED=false
set error=false

if [!flag!]==[start] goto wait_for_running
:begin
set error_count=
set /A error_count=0
for /F "usebackq delims=" %%i in (`DimQueryService %UTGID%/status`) do set STATE=%%i
echo %UTGID% state is !STATE!

if !STATE!==ERROR (
   echo Sending reset...
   DimSendCommand %UTGID% reset
   goto wait
)
if !STATE!==RUNNING (
   if [!STOPPED!]==[true] goto running
   echo Sending stop...
   DimSendCommand %UTGID% stop
   goto wait
)
if !STATE!==READY (
   if [!STOPPED!]==[true] goto send_start
   echo Sending reset...
   DimSendCommand %UTGID% reset
   goto wait
   :send_start
   echo Sending start...
   DimSendCommand %UTGID% start
   goto wait
)
if !STATE!==NOT_READY (
   set STOPPED=true
   if [!flag!]==[stop] (
      echo %UTGID% properly stopped
      goto stopped
   )
```

```
    echo Sending configure...
    DimSendCommand %UTGID% configure
    goto wait
)

:wait
sleep 1
set /A error_count=!error_count! + 1
if !error_count!==10 (
    echo %UTGID% can not be started
    goto error
)
for /F "usebackq delims=" %%i in (`DimQueryService %UTGID%/fsm_status`) do set
    FSM_STATE=%%i
rem echo !FSM_STATE!
if !FSM_STATE!==Executing goto wait
goto begin

:error
set error=true
echo Sending unload...
DimSendCommand %UTGID% unload
goto stopped

:wait_for_running
for /F "usebackq delims=" %%i in (`DimQueryService %UTGID%/status`) do set STATE
    =%%i
echo %UTGID% state is !STATE!
if !STATE!==RUNNING goto running
echo waiting...
sleep 2
goto wait_for_running

:running
echo %UTGID% properly started

:stopped
set UTGID=
set STATE=
set FSM_STATE=
set flag=
set STOPPED=
```

E.6 O *script* usado para iniciar os processos no sistema quase-*online*

E.6.1 daq_recv.sh

```
#!/bin/bash
echo "starting daq_recv"
. `dirname $0`/daq_recv_opts.sh

rename_logfiles
# if there are any

# set enviroment
source /home/nearonline/cmtuser/setup.sh
cd ${SITEROOT}/minerva/NEARONLINE/NEARONLINE_${RELEASEVERSION}/DaqRecv/${
    RELEASEVERSION}/cmt
source setup.sh

cd ${DATAPATH}

# shortcuts
export ONLINETASK='${GAUDIONLINEROOT}/${CMTCONFIG}/Gaudi.exe ${GAUDIONLINEROOT}/
    ${CMTCONFIG}/libGaudiOnline OnlineTask -auto'

# start DNS
${DIMROOT}/${CMTCONFIG}/DNS.exe &
echo $! > ${dns_pid}

# create a receiver buffer
echo "Starting reciever buffer"
export UTGID=InputBuffer
${GAUDIONLINEROOT}/${CMTCONFIG}/Gaudi.exe ${GAUDIONLINEROOT}/${CMTCONFIG}/
    libGaudiOnline OnlineTask -auto -main=${GAUDIONLINEROOT}/options/MBMfarm.opts
     -opt=${GAUDIONLINEROOT}/options/Daemon.opts -msgsvc=MessageSvc > ${
    inputbuffer_log} 2>&1 &
echo $! > ${inputbuffer_pid}

sleep 2

# start net receiver process
echo "Starting net reciever process"
export UTGID=NetRecv;
```

```
${GAUDIONLINEROOT}/${CMTCONFIG}/Gaudi.exe ${GAUDIONLINEROOT}/${CMTCONFIG}/
    libGaudiOnline OnlineTask -auto -main=${GAUDIONLINEROOT}/options/Main.opts -
    opt=${DAQRECVROOT}/options/MinervaNetRecv.opts -msgsvc=MessageSvc > ${
    netrecv_log} 2>&1 &
echo $! > ${netrecv_pid}

sleep 2

# start reformatter process
echo "Starting reformatter process"
export UTGID=Reformatter;
${GAUDIONLINEROOT}/${CMTCONFIG}/Gaudi.exe ${GAUDIONLINEROOT}/${CMTCONFIG}/
    libGaudiOnline OnlineTask -auto -main=${GAUDIONLINEROOT}/options/Main.opts -
    opt=${DAQRECVROOT}/options/Reformatter.opts -msgsvc=MessageSvc > ${
    reformatter_log} 2>&1 &

echo $! > ${reformatter_pid}
```

E.7 O arquivo de configuração para o processo NetSend

E.7.1 MinervaNetSend.opts

```
ApplicationMgr.ExtSvc          += { "IncidentSvc" };
ApplicationMgr.ExtSvc          += { "LHCb::MEPManager/MEPManager" };
ApplicationMgr.ExtSvc          += { "LHCb::OnlineEvtSelector/EventSelector"};

ApplicationMgr.SvcOptMapping   += { "LHCb::EventRunable/Runable" };
ApplicationMgr.SvcOptMapping   += { "LHCb::MEPManager/MEPManager" };
ApplicationMgr.SvcOptMapping   += { "LHCb::OnlineEvtSelector/EventSelector" };
ApplicationMgr.SvcOptMapping   += { "LHCb::RawDataCnvSvc/RawDataCnvSvc" };

ApplicationMgr.TopAlg          += {"LHCb::DimDataSender/Sender"};

Sender.DataSink                 = "NetRecv";
Sender.UseEventRequests         = false;
Sender.AllowSuspend             = true;
Sender.Connection               = "";
Sender.Compress                 = 0;
Sender.ChecksumType             = 1;
Sender.GenerateMD5              = false;
Sender.DataType                 = 2; // MDF_NONE=1, MDF_RECORDS=2, MDF_BANKS=3
Sender.BankLocation             = "LHCb::RawEventLocation::Default";

MessageSvc.LoggerOnly           = true;

ApplicationMgr.Runable          = "LHCb::EventRunable/Runable";

Runable.MEPManager              = "LHCb::MEPManager/MEPManager";
MEPManager.Buffers              = { "OUT" };
MEPManager.PartitionID          = 333;
MEPManager.PartitionBuffers     = true;
EventDataSvc.RootCLID           = 1;
EventDataSvc.ForceLeaves        = 1;
EventDataSvc.EnableFaultHandler = true ;
EventPersistencySvc.CnvServices += {"LHCb::RawDataCnvSvc/RawDataCnvSvc" };

EventSelector.Input             = "OUT";
EventSelector.Decode            = false;
EventSelector.REQ1              = "EvType=2;TriggerMask=0xffffffff,0xffffffff,0
    xffffffff,0xffffffff;VetoMask=0,0,0,0;MaskType=ANY;UserType=VIP;Frequency=
    PERC;Perc=100.0";
```

E.8 O arquivo de configuração para o processo NetRecv

E.8.1 MinervaNetRecv.opts

```
#include "$GAUDIONLINEROOT/options/OnlineEnvironment.opts"

ApplicationMgr.HistogramPersistency  = "NONE";
ApplicationMgr.EvtSel                = "NONE";
ApplicationMgr.Runable               = "LHCb::OnlineRunable/Runable";
ApplicationMgr.ExtSvc               += { "IncidentSvc" };
ApplicationMgr.ExtSvc               += { "LHCb::MEPManager/MEPManager" };
ApplicationMgr.SvcOptMapping        += { "LHCb::MEPManager/MEPManager" };
ApplicationMgr.SvcOptMapping        += { "LHCb::OnlineRunable/Runable" };
ApplicationMgr.SvcOptMapping        += { "LHCb::RawDataCnvSvc/RawDataCnvSvc" };
ApplicationMgr.SvcOptMapping        += { "LHCb::OnlineEvtSelector/EventSelector"
    };

Runable.Wait                         = 3;  // noop

MEPManager.Buffers                   = { "Event" };
MEPManager.PartitionID               = 333;
//@OnlineEnv.PartitionID;
MEPManager.PartitionBuffers          = false;

ApplicationMgr.ExtSvc               += { "LHCb::DimDataReceiver/Receiver" };
ApplicationMgr.SvcOptMapping        += { "LHCb::DimDataReceiver/Receiver" };

Receiver.Buffer                      = "Event";
Receiver.UseEventRequests            = false;
Receiver.DeclareAsynchonously        = false;
Receiver.MonitorSvcType              = @OnlineEnv.MonitorSvcType;

MessageSvc.OutputLevel               = 1;
```

E.9 O arquivo de configuração para o processo Reformatter

E.9.1 Reformatter.opts

```
ApplicationMgr.DLLs                 += { "MinervaEvent" };
ApplicationMgr.DLLs                 += { "ForestTest" };
AplicationMgr.DLLs                  += { "RawToDigit"};
ApplicationMgr.DLLs                 += { "DSTWriter"};
ApplicationMgr.DLLs                 += { "Gaucho" };

ApplicationMgr.ExtSvc               += { "MonitorSvc/MonitorSvc" };
ApplicationMgr.ExtSvc               += { "IncidentSvc" };
ApplicationMgr.ExtSvc               += { "LHCb::MEPManager/MEPManager" };
ApplicationMgr.ExtSvc               += { "LHCb::OnlineEvtSelector/EventSelector
    "};

ApplicationMgr.SvcOptMapping        += { "LHCb::EventRunable/Runable" };
ApplicationMgr.SvcOptMapping        += { "LHCb::MEPManager/MEPManager" };
ApplicationMgr.SvcOptMapping        += { "LHCb::OnlineEvtSelector/EventSelector"
     };
ApplicationMgr.SvcOptMapping        += { "LHCb::RawDataCnvSvc/RawDataCnvSvc" };
ApplicationMgr.SvcOptMapping        += { "PoolDbCacheSvc" };
ApplicationMgr.SvcOptMapping        += { "PoolDbCnvSvc/PoolRootEvtCnvSvc" };
ApplicationMgr.SvcOptMapping        += { "PoolDbCnvSvc/PoolRootTreeEvtCnvSvc" };

ApplicationMgr.TopAlg                = { "Minerva::DecodeRawEvent/Reformatter"
    };
ApplicationMgr.TopAlg               += { "BasicHistos" };

Reformatter.OutputLevel              = 4;
Reformatter.DecodeDiscriminators     = true;
BasicHistos.OutputLevel               = 4;
//BasicHistos.HistoProduce              = true;
//BasicHistos.HistoPrint                = true;
BasicHistos.MonitorHistograms         = false;
//BasicHistos.Do_FEB_Cal                = true;
BasicHistos.Do_Disc_Plots             = true;

ApplicationMgr.Runable               = "LHCb::EventRunable/Runable";

Runable.MEPManager                   = "LHCb::MEPManager/MEPManager";
```

```
MEPManager.Buffers                    = { "Event" };
MEPManager.PartitionID                = 333;
MEPManager.PartitionBuffers           = false;

EventDataSvc.RootCLID                 = 1;
EventDataSvc.ForceLeaves              = 1;
EventDataSvc.EnableFaultHandler       = true ;
EventPersistencySvc.CnvServices      += {"LHCb::RawDataCnvSvc/RawDataCnvSvc" };

EventSelector.Input                   = "Event";
EventSelector.Decode                  = false;
EventSelector.REQ1                    = "EvType=2;TriggerMask=0xffffffff,0
    xffffffff,0xffffffff,0xffffffff;VetoMask=0,0,0,0;MaskType=ANY;UserType=VIP;
    Frequency=PERC;Perc=100.0";

// load dictionaries
// #include "$STDOPTS/SimDicts.opts"
#include "$GAUDIPOOLDBROOT/options/GaudiPoolDbRoot.opts"
PoolDbCacheSvc.Dlls                  += { "DAQEventDict" };
PoolDbCacheSvc.Dlls                  += { "MinervaEventDict" };
PoolDbCacheSvc.Dlls                  += { "MinervaKernelDict" };

// write undecoded event into a file
ApplicationMgr.OutStream              = { "RawDigit" };
RawDigit.ItemList                     = { "/Event#1"
                                        , "/Event/DAQ#1"
                                        , "/Event/DAQ/Header#1"
                                        , "/Event/DAQ/RawDigits#1"};

OnlineEnv.RawDigitsFile               = "DATAFILE='PFN:UnpackedData.root' TYP='
    POOL_ROOTTREE' OPT='RECREATE'";
OnlineEnv.RawHistosFile               = "basic_histos.root";
OnlineEnv.DSTFile                     = {"FILE1 DATAFILE='DST.root' TYP='ROOT'
    OPT='NEW'"};

#include "/data_disk/rawdata/Output.opts"
RawDigit.Output                       = @OnlineEnv.RawDigitsFile;

// write histograms into a file
ApplicationMgr.HistogramPersistency   = "ROOT";
HistogramPersistencySvc.Outputfile    = @OnlineEnv.RawHistosFile;
```

E.10 A classe ConvertChannelID

E.10.1 ConvertChannelID.h

```
#ifndef CONVERTCHANNELID_H
#define CONVERTCHANNELID_H

#include <string>

#include "Event/RawDigit.h"

// Convert the channelID in a 2D point
std::pair<int, int>* ChannelIDtoXY( Minerva::ChannelID );

// Convert the channelID in a 1D point
int ChannelIDtoX( Minerva::ChannelID );

#endif
```

E.10.2 ConvertChannelID.cpp

```
#include "ConvertChannelID.h"

// Convert the channelID into a 2D point
std::pair<int, int>* ChannelIDtoXY( Minerva::ChannelID channelID ) {

  // Parameters
  //int crocbegin  = 1;
  int chainbegin = 0;
  int boardbegin = 1;
  int pixelbegin = 0;

  int chainspercroc  = 4;
  int boardsperchain = 10;
  int pixelsperboard = 64;
  int pixelsperside  = 8;

  // Read information from channelID
  /*int link  = channelID.link();
    int crate = channelID.crate();
    int croc  = channelID.croc();*/
  int chain = channelID.chain();
  int board = channelID.board();
  int pixel = channelID.pixel();
  //int hit   = channelID.hit();

  // Assuming the following map of pixel number in a board
  /*
    56 57 58 59 60 61 62 63
    48 49 50 51 52 53 54 55
    40 41 42 43 44 45 46 47
    32 33 34 35 36 37 38 39
    24 25 26 27 28 29 30 31
    16 17 18 19 20 21 22 23
    8  9 10 11 12 13 14 15
    0  1  2  3  4  5  6  7
  */

  // Calculate position from pixel number
  int colum = pixel % pixelsperside;
  int row  = pixel / pixelsperside;

  int x = (board-boardbegin)*pixelsperside + colum;
```

```
  int y = (chain-chainbegin)*pixelsperside + row;

  std::pair<int,int>* XY = new std::pair<int,int>( x,y );

  return XY;

}

// Convert the channelID into a 1D point
int ChannelIDtoX( Minerva::ChannelID channelID ) {

  // Parameters
  //int crocbegin  = 1;
  int chainbegin = 0;
  int boardbegin = 1;
  int pixelbegin = 0;

  int chainspercroc  = 4;
  int boardsperchain = 10;
  int pixelsperboard = 64;

  // Read information from channelID
  /*int link  = channelID.link();
    int crate = channelID.crate();
    int croc  = channelID.croc();*/
  int chain = channelID.chain();
  int board = channelID.board();
  int pixel = channelID.pixel();
  //int hit   = channelID.hit();

  int x = (chain-chainbegin)*boardsperchain*pixelsperboard + (board-boardbegin)*
      pixelsperboard + pixel;

  return x;

}
```

E.11 A classe HitCountsAlg

E.11.1 HitCountsAlg.h

```
//#############################################################
//
// This algorithm creates histograms from SupDigits
// They show how many times a pixel has received a signal
// outside the pedestal distribution during a run.
// It shows the same information in 2D and 1D histograms
//
//#############################################################

#ifndef HITCOUNTSALG_H
#define HITCOUNTSALG_H

#include <string>
#include "Event/RawDigit.h"
#include "GaudiAlg/GaudiHistoAlg.h"

class HitCountsAlg : public GaudiHistoAlg {

 public:
  HitCountsAlg(const std::string& name, ISvcLocator* pSvcLocator);
  virtual StatusCode initialize();
  virtual StatusCode execute();
  virtual StatusCode finalize();
  virtual ~HitCountsAlg() {};

 private:
  int m_nPixels;
  int m_nSidePixels;
  int m_nBoards;
  int m_nChains;
  int m_nCrocs;
  int m_crocNumber;
  std::string m_digitLocation;
  AIDA::IHistogram1D* m_histo1D;
  AIDA::IHistogram2D* m_histo2D;

}; // end of class

#endif
```

E.11.2 HitCountsAlg.cpp

```
#include "HitCountsAlg.h"
#include "ConvertChannelID.h"
#include "GaudiKernel/DeclareFactoryEntries.h"
#include "GaudiKernel/AlgFactory.h"

DECLARE_ALGORITHM_FACTORY( HitCountsAlg );

HitCountsAlg::HitCountsAlg(const std::string& name, ISvcLocator* pSvcLocator)
  : GaudiHistoAlg(name, pSvcLocator)
  , m_nPixels(64)
  , m_nSidePixels(8)
  , m_nBoards(10)
  , m_nChains(4) {

  declareProperty("CrocNumber", m_crocNumber = 1);
  declareProperty("DigitLocation", m_digitLocation = Minerva::RawDigitLocation::
      Suppressed);

}

StatusCode HitCountsAlg::initialize() {
  StatusCode sc = GaudiHistoAlg::initialize(); // must be executed first
  if ( sc.isFailure() ) return sc;  // error printed already by GaudiHistAlg

  info() << "==> Initialize" << endreq;

  // Booking histograms
  int lowX     = 0;
  int highX    = m_nChains*m_nBoards*m_nPixels;
  int binsX    = highX;
  int low2DX   = 0;
  int high2DX  = m_nBoards*m_nSidePixels;
  int bins2DX  = high2DX;
  int low2DY   = 0;
  int high2DY  = m_nChains*m_nSidePixels;
  int bins2DY  = high2DY;
  int lowGain  = 0;
  int highGain = 3500;
  int binsGain = 350;

  m_histo1D = book1D( "Hit_Count1D","Hit_Count1D",lowX,highX,binsX );
```

```
  m_histo2D = book2D( "Hit_Count2D","Hit_Count2D",low2DX,high2DX,bins2DX,low2DY,
      high2DY,bins2DY );

  return StatusCode::SUCCESS;
}

StatusCode HitCountsAlg::execute() {

  debug() << "==> Execute" << endreq;

  // Get RawDigits from TES
  Minerva::RawDigits*
    rawdigits = get<Minerva::RawDigits>(m_digitLocation);

  // Get event information from each digit in RawDigits (ndigits digits readed)
  int ndigits = 0;
  for ( Minerva::RawDigits::iterator it = rawdigits->begin();
        it != rawdigits->end();
        ++it ) {

    // The iterator is a pointer to a digit
    Minerva::RawDigit* digit = *it;

    // Get the channel ID from the digit
    Minerva::ChannelID channelID = digit -> channelID();

    // Only pixels in the specified croc will be used
    if ( channelID.croc() == m_crocNumber ) {

      // Transform the channelID to use it in histogram axes
      int x = ChannelIDtoX( channelID );
      std::pair<int,int>* XY = ChannelIDtoXY( channelID );

      // Fill histograms with data
      fill( m_histo1D, x, 1, "Hit_Count1D" );
      fill( m_histo2D, XY -> first, XY -> second, 1, "Hit_Count2D" );

      // Show some information for debugging
      debug() << "croc    = " << channelID.croc()  << endreq;
      debug() << "chain   = " << channelID.chain() << endreq;
      debug() << "board   = " << channelID.board() << endreq;
      debug() << "pixel   = " << channelID.pixel() << endreq;
      debug() << "X       = " << XY -> first       << endreq;
      debug() << "Y       = " << XY -> second      << endreq;
```

```
      debug() << "x       = " << x                    << endreq;

      // One digit has been read
      ++ndigits;

    } // end if

  } // end for

  return StatusCode::SUCCESS;
}

StatusCode HitCountsAlg::finalize() {

  info() << "==> Finalize" << endreq;

  return GaudiHistoAlg::finalize();  // must be called after all other actions
}
```

E.12 A classe ADCCountsAlg

E.12.1 ADCCountsAlg.h

```
//#########################################################################
// This algorithm creates histograms from RawDigits.
// They show "ADCCounts" and "Mean ADCCounts" for each pixel
// in a specified croc for the 3 avaliable gains (low, med, hi)
//#########################################################################

#ifndef ADCCOUNTSALG_H
#define ADCCOUNTSALG_H

#include <string>
#include "Event/RawDigit.h"
#include "GaudiAlg/GaudiHistoAlg.h"

class ADCCountsAlg : public GaudiHistoAlg {

 public:
  ADCCountsAlg(const std::string& name, ISvcLocator* pSvcLocator);
  virtual StatusCode initialize();
  virtual StatusCode execute();
  virtual StatusCode finalize();
  virtual ~ADCCountsAlg() {};
 private:
  int m_nPixels;
  int m_nSidePixels;
  int m_nBoards;
  int m_nChains;
  int m_crocNumber;
  std::string m_digitLocation;
  AIDA::IHistogram2D* m_histo2D_01;
  AIDA::IHistogram2D* m_histo2D_02;
  AIDA::IHistogram2D* m_histo2D_03;
  AIDA::IProfile1D*   m_profile_01;
  AIDA::IProfile1D*   m_profile_02;
  AIDA::IProfile1D*   m_profile_03;

};// end of class

#endif
```

E.12.2 ADCCountsAlg.cpp

```
#include "ADCCountsAlg.h"
#include "ConvertChannelID.h"
#include "GaudiKernel/DeclareFactoryEntries.h"
#include "GaudiKernel/AlgFactory.h"

DECLARE_ALGORITHM_FACTORY( ADCCountsAlg );

ADCCountsAlg::ADCCountsAlg(const std::string& name, ISvcLocator* pSvcLocator)
  : GaudiHistoAlg(name, pSvcLocator)
  , m_nPixels(64)
  , m_nSidePixels(8)
  , m_nBoards(10)
  , m_nChains(4) {

  declareProperty("CrocNumber", m_crocNumber = 1);
  declareProperty("DigitLocation", m_digitLocation = Minerva::RawDigitLocation::
      Default);

}

StatusCode ADCCountsAlg::initialize() {

  StatusCode sc = GaudiHistoAlg::initialize(); // must be executed first
  if ( sc.isFailure() ) return sc;  // error printed already by GaudiHistAlg

  info() << "==> Initialize" << endreq;

  // Booking histograms
  int lowX     = 0;
  int highX    = m_nChains*m_nBoards*m_nPixels;
  int binsX    = highX;
  int low2DX   = 0;
  int high2DX  = m_nBoards*m_nSidePixels;
  int bins2DX  = high2DX;
  int low2DY   = 0;
  int high2DY  = m_nChains*m_nSidePixels;
  int bins2DY  = high2DY;
  int lowGain  = 0;
  int highGain = 3500;
  int binsGain = 350;

```

```
  m_histo2D_01 = book2D( "Low_Gain","Low_Gain",lowX,highX,binsX,lowGain,highGain
      ,binsGain );
  m_histo2D_02 = book2D( "Medium_Gain","Medium_Gain",lowX,highX,binsX,lowGain,
      highGain,binsGain );
  m_histo2D_03 = book2D( "High_Gain","High_Gain",lowX,highX,binsX,lowGain,
      highGain,binsGain );
  m_profile_01 = bookProfile1D( "Low_Gain_Mean","Low_Gain_Mean",lowX,highX,binsX
      );
  m_profile_02 = bookProfile1D( "Medium_Gain_Mean","Medium_Gain_Mean",lowX,highX
      ,binsX);
  m_profile_03 = bookProfile1D( "High_Gain_Mean","High_Gain_Mean",lowX,highX,
      binsX);

  return StatusCode::SUCCESS;
}

StatusCode ADCCountsAlg::execute() {

  debug() << "==> Execute" << endreq;

  // Get RawDigits from TES
  Minerva::RawDigits*
    rawdigits = get<Minerva::RawDigits>(m_digitLocation);

  // Get event information from each digit in RawDigits (ndigits digits readed)
  int ndigits = 0;
  for ( Minerva::RawDigits::iterator it = rawdigits->begin();
        it != rawdigits->end();
        ++it ) {

    // The iterator is a pointer to a digit
    Minerva::RawDigit* digit = *it;

    // Get the channel ID from the digit
    Minerva::ChannelID channelID = digit -> channelID();

    // Only pixels in the specified croc will be used
    if ( channelID.croc() == m_crocNumber ) {

      // Transform the channelID to use it in histogram axes
      int x = ChannelIDtoX( channelID );

      // Get information from digit
      int qlo  = digit -> qlo();
```

```
        int qmed = digit -> qmed();
        int qhi  = digit -> qhi();

        // Fill histograms
        fill( m_histo2D_01, x, qlo,  1, "Low_Gain" );
        fill( m_histo2D_02, x, qmed, 1, "Medium_Gain" );
        fill( m_histo2D_03, x, qhi,  1, "High_Gain" );
        fill( m_profile_01, x, qlo,  1, "Low_Gain_Mean" );
        fill( m_profile_02, x, qmed, 1, "Medium_Gain_Mean" );
        fill( m_profile_03, x, qhi,  1, "High_Gain_Mean" );

        // Show some information for debugging
        debug() << "croc   = " << channelID.croc()  << endreq;
        debug() << "chain  = " << channelID.chain() << endreq;
        debug() << "board  = " << channelID.board() << endreq;
        debug() << "pixel  = " << channelID.pixel() << endreq;
        debug() << "x      = " << x                 << endreq;

        // One digit has been read
        ++ndigits;

      } // end if

    } // end for

    return StatusCode::SUCCESS;
}

StatusCode ADCCountsAlg::finalize() {

    info() << "==> Finalize" << endreq;

    return GaudiHistoAlg::finalize();  // must be called after all other actions
}
```

Bibliografia

[1] Karagiorgi, G.A. Aguilar-Arevalo, J.M. Conrad, and M.H. Shaevitz. Leptonic CP violation studies at MiniBooNE in the (3+2) sterile neutrino oscillation hypothesis, Phys Rev D 75 (2007)1.

[2] Z. Maki, M. Nakagawa, and S. Sakata, Prog. Theor. Phys. 28, 870 (1962).

[3] von Bayer, O. Hahn, L. Meitner, Phys. Zeitschrift, 12, January, 1911, p. 378

[4] C.D. Ellis, B.A. Wooster, The average energy of desintegration of Radium E,Proc. Roy. Soc. A117(1927) 109-123

[5] W. Pauli, carta enviada à Conferência de Tubingen, Dec. 1930.

[6] J. Chadwick, Possible existence of a neutron, Nature 129 (1932)12.

[7] E. Fermi, Ricercha Scient.**2** 12(1933).

[8] F.Perrin, Comptes rendues **197**, 1624(1933).

[9] E. Fermi, Z. Physik, **88** 161(1934).

[10] F. Reines and C.L. Cowan, The neutrino, Nature 178 (1956)446.

[11] M. Goldhaber, L. Grodzins, and A.W. Sunyar, Helicity of neutrinos, Phys. Rev. 109 (1958)1015.

[12] R. Davis and D.S. Harmer, Attempt to Observe the $Cl^37(\overline{\nu},e^-)Ar^37$ Reaction Induced by Reactor Antineutrinos, Bull. Am. Phys. Soc. 4, (1959)217.

[13] G. Danby et al., Observation of high-energy neutrino reactions and the existence of two kinds of neutrinos, Phys. Rev. Lett. 9 (1962)36.

[14] Gargamelle Neutrino Collaboration: F.J. Hasert et al., Observation of neutrino-like interactions without muon or electron in the Gargamelle neutrino experiment, Phys. Lett. B **46** (1973)138.

[15] R. Davis,D.S. Harner, and K.C. Hoffman, Search for neutrinos from the sun, Phys. Rev. Lett. 20 (1968)1205.

[16] V.N. Gribov and B. Pontecorvo, Phys. Lett B **28**(1969)493.

[17] ALEPH Collaboration:D. Decamp et al., Determination of the Number of Light Neutrino Species, Phys. Lett. B **231**(1989)519.

[18] Delphi Collaboration:P.A. Aarnio et al.,Measurement of the Mass and Width of the Z0 Paricle from Multi-Hadronic Final States Produced in the e+ e- Annihilation, Phys. Lett. B **231** (1989)539.

[19] L3 Collaboration:B. Adeva et al.,A Determination of the Properties of the Neutral Intermediate Vector Boson Z0, phys. Lett. B **231** (1989)509.

[20] OPAL Collaboration:M.Z. Akrawy et al.,Measurement of the Z0 Mass and Width with the OPAL Detector at LEP, Phys. Lett. B bf 231 (1989)530.

[21] DONUT COllaboration: T. Patzak, First direct observation of the tau neutrino, Europhys. News 32, 56 (2001).

[22] C.S. Wu et al., Phys. Rev. **105** (1957)1413.

[23] Super-Kamiokande Collaboration:Y. Fukuda et al., Evidence for oscillation of atmospheric neutrinos, Phys. Rev. Lett. 81, (1998)1562.

[24] Soudan 2 Collaboration:M.C. Sanchez et al., Observation of atmospheric neutrino oscillations in Soudan 2, Phys. Rev. D **68** (2003)113004.

[25] SNO Collaboration:Q.R. Ahmad et al.,Direct evidence for neutrino flavor transformation from neutral-current interactions in the Sudbury Neutrino Observatory, Phys. Rev. Lett. 89, (2002)01131.

[26] R.N. Mohapatra e P.B. Pal, Massive Neutrinos in Physics and Astrophysics, World Scientific (1998)

[27] KamLAND Collaboration:K. Eguchi et al.,First results from KamLAND:Evidence for reactor anti-neutrino disappearance, Phys. Rev. Lett. 90, (2003)021802.

[28] K2K Collaboration: M.H. Ahn et al.,Measurement of neutrino oscillation by the K2K experiment, Phys. Rev. D **74** (2006)072003.

[29] MINOS Collaboration:D.G. Michael et al., Observation of Muon Neutrino Disappearance with the MINOS Detector in the NuMI Neutrino Beam, Phys. Rev. Lett. 97 (2006)191801.

[30] M. Apollonio et al., Search for neutrino oscillations on a long base-line at the CHOOZ nuclear power station, Eur. Phys. J. C27, 331-374 (2003).

[31] Particle Data Group, J. Phys. G: Nucl. Part. Phys. **33** (2006).

[32] J. Hylen et al., NuMI Technical Design Handbook, Internal NuMI report (2003).

[33] R. M. Zwaska, Accelerator Systems and Instrumentation for the NuMI Neutrino Beam, PhD thesis University of Texas at Austin, 2005.

[34] Proposal to Perform a High-Statistics Neutrino Scattering Experiment Using a Fine-grained Detector in the NuMI Beam, http://minerva.fnal.gov/proposal.pdf

[35] H. Kluttig, J.G. Morfin e W.Van Doninck, Phys.Lett. B**71** (1977),446.

[36] O. Erriquez et al., Phys. Lett. B **73** (1978)350.

[37] W. Krenz et al.,Nucl. Phys. B**135** (1978)45.

[38] M. Pohl et al., Phys. Lett. B**82** (1979)461.

[39] W.-Y. Lee et al.,Phys. Rev. Lett. **38** (1977)202.

[40] S.J. Barrish et al., Phys. Rev. Lett **33** (1974)448.

[41] M. Derrick et al., Phys. Lett B **92** (1980)363.

[42] M. Derrick et al., Phys. Lett B **23** (1981)569.

[43] N.J. Baker et al., Phys. Rev. D **23** (1981)2495.

[44] C. Arnault, "CMT: A software configuration management tool", Computing in high energy and nuclear physics (CHEP 2000) 692-695 , Padova, Italy.

[45] C. Arnault, "Configuration Management Tool, CMT Manual", http://www.cmtsite.org/.

[46] Worldwide LHC Computing Grid (WLCG)'s collaboration "LCG Project - Applications Area", http://lcgapp.cern.ch/

[47] G. Barrand et al., GAUDI: A Software Architecture and Framework for building HEP Data Processing Applications, CHEP 2000 proceedings, Padova, Feb. 2000.

[48] http://cern.ch/Gaudi

[49] Gloria Corti et al., "Software for the LHCb Experiment", IEEE TRANSACTIONS ON NUCLEAR SCIENCE, VOL. 53, NO. 3, JUNE 2006.

[50] http://lhcb-comp.web.cern.ch/lhcb-comp/

[51] A colaboração MINERνA , "MINERvA Technical design Report", MINERνA Document Database #700-v28, http://minerva-docdb.fnal.gov/, Feb 2008.

[52] C. Gaspar et al., DIM, a Portable, Light Weight Package for Information Publishing, Data Transfer and Inter-process Communication, CHEP 2000, Padova, Italy

[53] E. van Herwijnen, Control and Monitoring of on-line trigger algorithms using Gaucho, CHEP 2006, Mumbai, India, 2006

[54] M.Frank et. al., The LHCb High Level Trigger Infrastructure, Journal of Physics: Conference Series 119 (2008) 022023, International Conference on Computing in High Energy and Nuclear Physics (CHEP07)

[55] F. Alessio et. al., LHCb Online event processing and filtering, Journal of Physics: Conference Series 119 (2008) 022003, International Conference on Computing in High Energy and Nuclear Physics (CHEP07)

[56] A. Fiorentini "Minervas Near Online Manual", MINERVA Document Database #32347-v2, http://minerva-docdb.fnal.gov, February 2009

[57] B. Baldin "MINERVA Chain Readout Controller", MINERVA Document Database #1516-v1, http://minerva-docdb.fnal.gov, março 2007

[58] B. Baldin "MINERVA CROC Interface Module", MINERVA Document Database #1238-v3, http://minerva-docdb.fnal.gov, julho 2008

[59] E.H. Bellamy et al., "Absolute calibration and monitoring of a spectrometric channel using a photomultiplier", Nuclear Instruments and Methods in Physics Research A 339 (1994) 468-476.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo