Encuentro de Investigadores y Docentes de Ingeniería 2008
EnIDI 2008, Mendoza, Argentina

Sistema Monitor de Funcionamiento para los Detectores de Fluorescencia del Observatorio Pierre Auger

P. Caif, J. Gitto, B. Sanchez

pcaif@frm.utn.edu.ar, jgitto@frm.utn.edu.ar, bsanchez@frm.utn.edu.ar
Laboratorio Pierre Auger, Facultad Regional Mendoza, Universidad Tecnológica Nacional

RESUMEN: Se presenta una síntesis global de las técnicas y componentes informáticos desarrollados para el sistema de monitoreo dentro de un entorno de programación complejo que requiere codificación precisa y validación en el sitio del Observatorio Pierre Auger. El monitoreo del funcionamiento de los detectores de fluorescencia es una tarea concurrente a las actividades de adquisición de datos primaria. Su objetivo es la permanente vigilancia de los parámetros del soporte físico y lógico de los telescopios y su comparación con valores de referencia. El sistema monitor ha de permitir el almacenamiento de los parámetros de operación leídos, mensajes de error y advertencias para su visualización en línea o recuperación para análisis posteriores.

Palabras clave: sistema – monitoreo – funcionamiento – fluorescencia - programación

ABSTRACT: A synthesis is exposed to give a glance of the computer software components developed for the performance monitoring system within a complex programming framework that requires precise codification and validation in the site of the Pierre Auger Observatory. The fluorescence detectors performance monitoring is a concurrent task to the primary data acquisition activities. Its objective is the permanent monitoring of the hardware and software telescope parameters and its comparison with reference values. The monitoring software must provide capabilities to store the read operation parameters, error and warning messages that would to allow on-line display or Its recovering for later analysis.

Key Words: system – monitoring – performance – fluorescence -software

1 OBJETIVOS

El objetivo de este proyecto es desarrollar un prototipo de sistema informático para monitorear el funcionamiento de los telescopios de fluorescencia (FD) del PAO [1] y su validación en los sitios de desarrollo y emplazamiento final en Alemania y Malargue.

1.1 *Objetivos particulares*

Definir las metodologías de implementación del código de programación realizando:

- Evaluación y elección de los entornos y ambientes de programación en SO Linux que posibiliten la integración con el marco de trabajo de Auger.
- Generación y optimización de códigos de programas para su integración y operación efectiva en los telescopios.
- Aplicar técnicas de conversión de programas en procesos demonios e integración con rutinas de E/S mediante protocolo TCP/IP.
- Crear un método de acceso estándar a los programas de monitoreo de cada telescopio destinado a terceras aplicaciones.

1.2 *Antecedentes*

La necesidad de monitoreo sistemático de los parámetros de los detectores de fluorescencia correspondientes a los telescopios, las cámaras, foto-tubos y sistema de control de variables lentas, ha sido considerada como tarea desde Marzo del 2000. Detalles más específicos del sistema a implementar fueron acordados en reuniones de trabajo durante Agosto del 2003 en Karlsruhe [2] y posteriores que se ajustan al reporte de diseño técnico (TDR) [3], del proyecto Pierre Auger. Estos marcan las pautas principales de este desarrollo y tareas realizadas por el grupo del laboratorio Auger UTN, Mendoza, en colaboración con los grupos del IK/IPE, Forschungszentrum Karlsruhe, Alemania y la Università degli Studi di Milano & INFN, Italia.

2 DESCRIPCIÓN GLOBAL

2.1 *Descripción del soporte físico, telescopios*

Los detectores de fluorescencia del observatorio Pierre Auger están conformados por cuatro complejos llamados “ojos”, los cuales contienen seis telescopios de fluorescencia de 30º x 30º de apertura cada uno para cubrir una área celeste de 180º x 30º entre los seis de cada grupo. La organización de la electrónica sigue la estructura geométrica del detector. A su vez, cada telescopio llamado “espejo” se compone de una cámara con 440 foto-tubos (PMTs) que se interconectan con un sub-gabinete de electrónica y una computadora denominada “pc-espejo” que realiza la lectura primaria de datos. Cada sub-gabinete contiene 20 placas disparadoras de 1er nivel (FLT) y una placa disparadora de segundo nivel (SLT). El sistema completo consiste en 24 telescopios de óptica Schmidt con 10560 canales para lectura, distribuidos en cuatro estaciones u ojos detectores de fluorescencia. La tarea de supervisión y lectura de parámetros es realizada por medio de computadoras industriales, bajo el sistema operativo Linux. Las señales provenientes de los PMTs, debidamente conformadas, son digitalizadas y almacenadas mediante un proceso basado en técnicas de disparos que permiten iniciar la lectura de los pixeles o imágenes de la cámara, ante la ocurrencia de un evento.

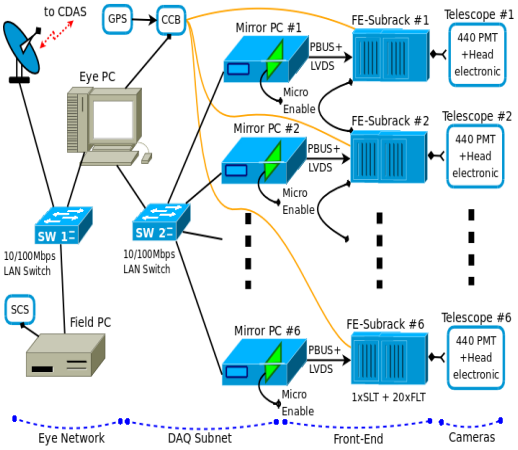

Figura 1. Arquitectura del soporte físico correspondientes a un ojo, red de comunicaciones

Finalmente, una computadora de red (pc-ojo), comprime los datos, mejora la decisión del disparo, obtiene los parámetros de un mismo evento y los transfiere a las instalaciones del centro de adquisición de datos principal (CDAS).
Todo el complejo es operado remotamente mediante un sistema de control de variables lentas (SCS) que controla el encendido, apagado, apertura de compuertas y demás dispositivos de cada telescopio. En la Fig. 1, puede observarse el esquema general.

2.2 *El sistema de monitoreo, soporte lógico*

El sistema monitor está estructurado por un conjunto de cuatro programas: MiMoD, EyeMoD, StaMoD y FDMonGUI. Los tres primeros, son procesos demonios y conforman el núcleo del sistema, mientras que el restante es un programa de aplicación para usuarios (UI/GUI), necesario para verificar el funcionamiento y visualizar los resultados .
La Fig. 2 muestra el concepto global desde el punto de vista de los programas que componen el sistema; este esquema sólo muestra los principales componentes informáticos del mismo.

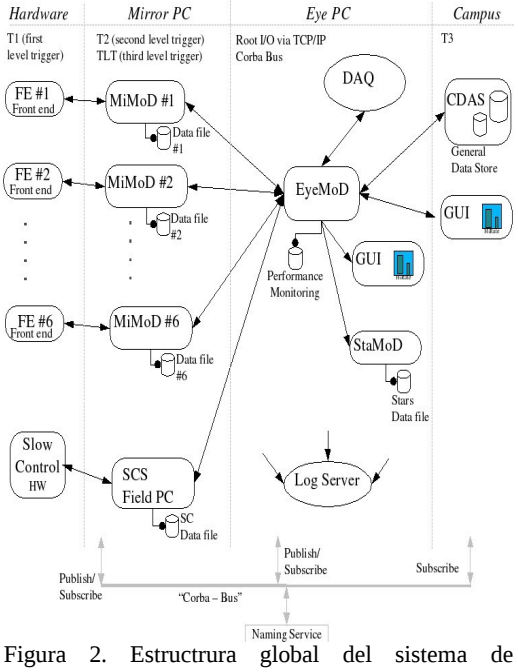

Figura 2. Estructrura global del sistema de monitoreo, programas componentes.

2.3 *Descripción de los programas, interacción*

Según premisas de diseño, cada uno de los MiMoD adquiere datos directamente de la electrónica conectada al respectivo telescopio, los compara con valores de referencia y almacena los datos en una memoria temporal de 500 palabras. EyeMoD colecta periódicamente datos del sistema de control (SCS), del sistema de adquisición de datos (DAQ) y se conecta secuenciadamente a cada MiMoD. Los datos obtenidos son almacenados en una memoria temporal de 2000 palabras. El demonio StaMoD calcula la posición de estrellas visibles que serán "vistas" por las cámaras y marca los píxeles brillantes que serán excluidos del resultado de la comparación que hace MiMoD. Por último, los módulos UI/GUI son los programas encargados de verificar el funcionamiento de los procesos del sistema y presentar la información.

3 METODOLOGÍAS PARA LA IMPLEMENTACIÓN DEL SISTEMA MONITOR.

En primera instancia, se determinaron entornos de programación, lugar y tipo de repositorios para el trabajo colaborativo, utilizando para este propósito el sistema de control de versiones CVS [4].
Se utilizó como lenguaje base de programación el C++ [5], el ambiente de desarrollo intregrado Kdevelop [6], bibliotecas QT [7] para el desarrollo de aplicaciones gráficas multilataforma asi como el conjunto de programas y librerias desarrolladas para acceder a la electrónica de los FD de Auger[8].
Para generar la documentación se utilizó el programa "Doxigen" [9] que permite crear la documentación a parir del cógido fuente de un programa o bien de un proyecto completo de programación. Doxigen permite la generación de la documentación en los formatos HTML, Latex, MS RTF, PostScrip, PDF y páginas del manual de Unix.
Como técnica de programación general, se decidió trabajar con el modelo orientado a objetos, debido a la naturaleza del desarrollo y los beneficios que provee dicha metodología. A saber: orden en el diseño de la aplicación, encapsulamiento de métodos, ocultamiento, abstracción, polimorfismo, herencia y modularización entre otros.

3.1 *El marco de trabajo de Auger, metodología para el intercambio de datos*

La comunicación entre los procesos, debe optimizarse para mantener el tráfico de red tan bajo como sea posible, evitando cualquier congestión que interfiera en el proceso principal de adquisición de datos (DAQ). Por este requisito, se estudiaron y utilizaron tres técnicas distintas para la comunicación e intercambio de datos; las mismas son:

C*ORBA* (*Common Object Request Broker Architecture*): es un mecanismo estándar que permite la interoperabilidad entre diferentes códigos de programas escritos en diferentes lenguajes y ejecutados en diferentes plataformas de sistemas operativos [10].

ROOT: es un entorno de desarrollo para el análisis de datos. Soporta la transferencia de estructuras de datos polimórficos arbitrariamente complejas. Se usa para el intercambio de grandes volúmenes de datos sobre la red [11].
OPC (Object-Linking and Embedding for Process Control): es un estándar de comunicación de datos entre procesos industriales y el conjunto de los objetos, intefaces y métodos, que facilita la interoperabilidad con dispositivos, instrumentos, controladores y programas en plataformas MS Windows, Linux y otras [12].

4 DESARROLO Y PRODUCCIÓN DEL CÓDIGO.

En la práctica, el proyecto requirió desarrollar los siguientes módulos comunes a los programas principales:

- Módulo de memoria intermedia multidimencional de acceso aleatorio y refresco cíclico (RingBuffer).
- Módulo para la comunicación entre procesos vía red TCP/IP.
- Código para conversión de los programas a procesos demonios del sistema operativo (SO).
- Aplicación de inteface de usuario UI/GUI para ensayo, prueba y depuración del código.

4.1 *Módulo de memoria intermedia, estructura RingBuffer (RB).*

En este desarrollo se utilizaron conceptos de diseño orientado a objetos tales como constructores, herencia de clases y otros propios de este paradigma de programación.
Varios códigos fueron desarrollados para esta investigacion. La Fig. 3, muestra el concepto general en el que puede verse la estructura de la clase RB.
A continuación se hace una breve descripción de las partes que componen el módilo RB explicitando parte su estructura interna y la definición de los objetos, clases y métodos para este módulo.

4.1.1 *RingBufferServer*

Es un proceso que se encargarga de escuchar y proveer las solicitudes de petición de datos emitida por los clientes. Ejecuta los métodos correspondientes (heredados) de la clase RingBuffer para poder entregar los datos pedidos.

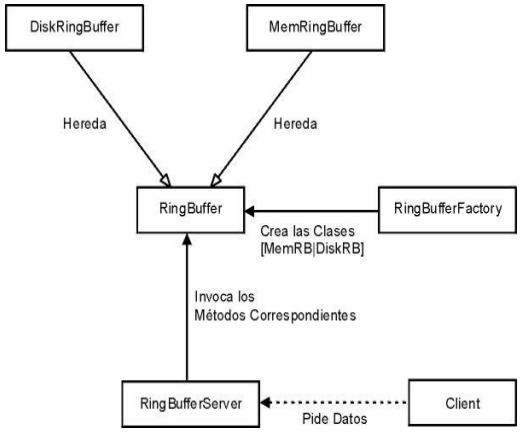

Figura 3. Estructura de la clase RingBuffer, principales componentes

4.1.2 *Clase RingBufferFactory*

Esta clase es la implementación del patrón "Factory" para la clase abstracta RingBuffer y las clases MemRingBuffer y DiskRingBuffer. Su función es dar visibilidad a los otros módulos creando las diferentes clases RBr: MemRingBuffer o , DiskRingBuffer, según el valor enviado como parámetro.

4.1.2.1 *Miembros de Datos*

- ✗ static const short MEM_RING_BUFFER=0: constante que identifica a la clase MemRingBuffer.
- ✗ *static const short DISK_RING_BUFFER=1: constante que identifica a la clase DiskRingBuffer.*

4.1.2.2 *Métodos públicos*

Constructores:
- ✗ RingBufferFactory ():

Destructores:
- ✗ ~RingBufferFactory():

Métodos:
- ✗ static RingBuffer * create(const short type, const char * fileName="", const char * mode="update")
  Se encargarga de crear los diferentes tipos de clases RingBuffer. Mediante el parámetro

"type" se especifica el tipo de matriz a crear. El parámetro fileName especifica la ruta del archivo que será accedido y mode determina el tipo de acceso(update, read, recreate).

4.1.3 *Definición de la clase RingBuffer*

Este componente, se define como una clase abstracta de las cuales heredan las clases MemRingBuffer y DiskRingBuffer. Contiene todos los métodos virtuales que se implementan en las clases subordinadas.

4.1.3.1 *Métodos públicos*

Constructores:
- RingBuffer()

Destructores:
- virtual ~RingBuffer()

4.1.3.2 *Métodos Virtuales*

Se definen a partir de los métodos implementados por las clases que a su vez heredan de RingBuffer. Ejemplos de estos métodos son los siguientes:

- virtual int writeLast(Matrix * Data)=0:
- virtual int writeGPSTime(Matrix * Data)=0:
- virtual int writePoint(Matrix * Data,const int nData)=0:
- virtual Matrix * readLast (void)=0:
- virtual Matrix * readGPSTime (int gpstime)=0:
- *virtual Matrix * readPoint (int nData)=0:*

4.1.4 *Clase DiskRingBuffer*

Su propósito es leer y almacenar los datos (objetos matrices), en un archivo. Para realizar éstas funciones utiliza la clase *Tmatrix* de Root. Esta clase guarda los datos de la matriz usando como llave o puntero el parámetro del telescopio llamado GPSTime. Como este último se trata de un número creciente y que no se repite, permite la posterior recuperación o modificación de datos sin ambiguedades. También utiliza otras clases pertenecientes a Root:

TFile: utilizada para crear un archivo root y obtener todas la funcionalidades que ofrece.
TList: utilizada para obtener un listado ordenado de las últimos 500/2000 registros que contiene el ringbuffer.
Tkey: utilizada para almacenar la llave de una matriz.

4.1.4.1 *Miembros de Datos*

- TFile *file: Es un puntero a un objeto TFile utilizado para mantener el archivo abierto y realizar las operaciones de lectura y escritura de matrix
- TList *keys: Mantiene una lista de las 500/2000 últimas matrices guardadas en el ringbuffer.

4.1.4.2 *Métodos públicos*

Constructores:
- DiskRingBuffer(const char * filename, const char * mode="update")
  Se encarga de abrir el archivo indicado en filename en modo "update" por defecto. Esto significa que si el archivo existe lo abre y se pueden agregar datos, si no lo crea. Otros modos pueden ser indicados en "mode" y son: "Read" (solo lectura) y "Recreate" (crea nuevamente el archivo borrando todos los datos)

Destructor:
- ~DiskRingBuffer(): cierra el archivo y libera la memoria.

4.1.5 *Clase MemRingBuffer*

Esta clase es la encargada de leer y alamcenar los objetos matrices en memoria. Para realizar éstas funciones utiliza la clase Tmatrix, Tlist y Tkey de Root. A ligual que la clase DiskRingBuffer, también utiliza como llave o puntero al parámetro GPSTime.

4.1.5.1. Métodos públicos

Constructores:
- MemRingBuffer(const char * filename, const char * mode): es el encargado de cargar en memoria el vector de las últimas 500/2000 matrices.

Destructor:
- ~MemRingBuffer(): libera el espacio de memoria utilizado por el arreglo de matrices.

4.1.6 *Client*

Este proceso es el "cliente" que realiza la petición de datos al servidor RingBufferServer. Los datos que pueden solicitar los clientes son:

- RawBufferRing: matriz de datos obtenidas del

FE en un gpstime específico, un puntero o la última matriz procesada.

- ✗ ResultBufferRing: matriz de resultados obtenidas de la comparación entre datos con valores de referencia y los rangos de cada parámetro(Variance, Pedestal, Threshold, etc.). Determinada por un gpstime específico, un puntero o la última matriz procesada.
- ✗ Write RawBufferRing: escritura de la matriz de datos obtenida por el FE en un gpstime específico, un puntero o la última matriz procesada.
- ✗ Write ResultBufferRing: escritura de la matriz de datos de resultado en un gpstime específico, un puntero o la última matriz procesada.

4.2 *Módulo de comunicaciones TCP/IP, Entrada y Salida (I/O)*

La comunicación entre los procesos demonios y otras aplicaciones de monitoreo como la UI o GUI se realiza mediante protocolo de red TCP/IP usando las clases *TServerSocket* y *TSocket* de las librerías de Root. A partir de estas se crearon dos clases adaptadas a este proyecto: *SocketAcceptor* y *SocketConnector*, estas últimas, heredan las propiedades y el comportamiento de las antes nombradas. Cada proceso demonio utiliza SocketAcceptor para atender las peticiones de los clientes, mismas que son enviadas encapsuladas en un objeto o "Mensaje"; en nuestro caso es un objeto de la clase RingBufferRequest. A su vez, esta clase se compone de dos sub-clases llamadas MatrixRequest y ElementRequest; necesarias para que SocketAcceptor identifique el tipo de objeto y lo procese apropiadamente, permitiendo de este modo, acceder a datos apuntados.
Los parámetros son leídos directamente desde los sub-sistemas de adquisición de datos (pc-espejo) y pueden contener los datos en bruro de la "variance", "pedestal", "threshold", "hitrate" y otros, por cada uno de los 440 píxeles o foto-tubos de cada cámara. En el caso del demonio MiMoD; este adquiere los datos de parámetros, los compara y almacena los resultados de las comparaciones en memoria o disco. Los otros demonios (EyeMoD, StarMoD) u aplicaciones (UI/GUI) acceden a los datos vía red de comunicaciones. La Fig. 4, muestra el criterio empleado.

4.2.1 *Funcionamiento, técnica de ejecución de procesos*

Como los procesos de lectura, comparación y el proceso de atención de pedidos de los clientes de red, son inherentemente paralelos, la ejecución del servidor SocketAcceptor, se implementó utilizando la técnica de hilos "thread", del núcleo de Linux. La ejecución en hilos, permite que SocketAcceptor, permanezca bloqueado a la espera de una conexión entrante sin afectar el flujo de ejecución normal del programa principal. Para la sincronización del acceso a los datos en memoria (RB), se utilizó los denominados "semáforos" del núcleo del SO.
De esta forma, la comparación y la atención de peticiones se ejecutan en paralelo.

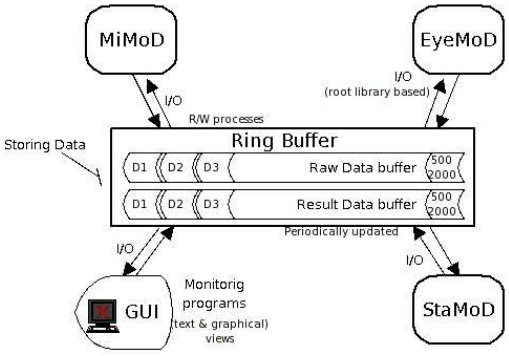

Figura 4. Diagrama de interaciones y comunicaciones entre los procesos

4.3 *Conversión de los programas a procesos demonios, ejecución en segundo plano*

Debido a que el programa de monitoreo realiza una lectura y comparación periódica (cada 16 segundos), fué conveniente convertir los programas en procesos demonios para que se ejecuten en segundo plano.
Un demonio es un proceso que no pertenece a ningún usuario y tiene como padre a INIT, el primer proceso que se ejecuta al iniciar un sistema Linux.
La Fig. 5, muestra la estructura en bloques del proceso demonio base desarrollado para MiMoD, EyeMoD y StaMoD.
En general, para convertir un programa a un proceso demonio, es necesario realizar el siguiente procedimiento [13] :

I. Se crea un hijo del proceso con la llamada del sistema "fork"; luego se bifurca o sale sale del proseso padre. De esta forma el proceso hijo se convierte en hijo de proceso INIT
II. Se crea una nueva sesión mediante la llamada al sistema "setsid", que no pertenece a ninguna terminal.

III. Se cierran los flujos de la entrada, la salida y el error estándar. Así los mensajes de salida del demonio no aparecen en la pantalla.
IV. Se establece el directorio actual coincidente con el directorio raiz del sistema.
V. Se registra el proceso como manejador de la señal SIGINT equivalente a "CTRL-c"
VI. Para evitar la salida abrupta del demonio y la posibilidad de dejar bloqueado el bus de la "pc-espejo", se atrapa la señal SINGINT, se procede a cerrar el SocketAcceptor, liberar la memoria del RB y finalmente, salir en forma ordenada del programa.

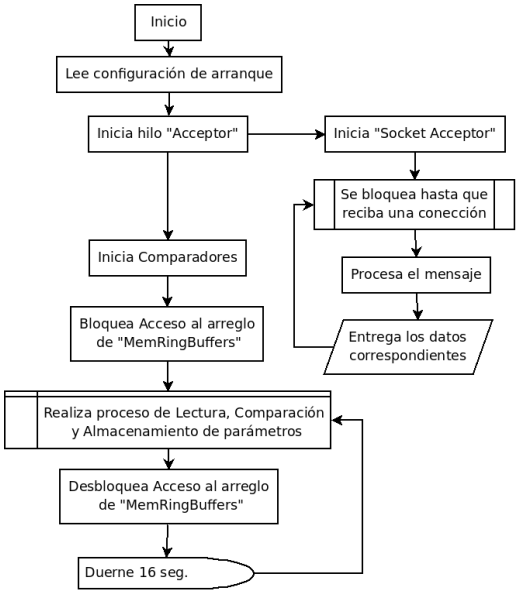

Figura 5. Estructura en bloques funcionales de los demonios

4.4 *Aplicación Interface de Usuario (UI/GUI)*

La interface de usuario preliminar (UI) fue desarrollada para efectuar las pruebas y ensayos de funcionamiento del sistema de monitor.
La UI fue programada en lenguaje C++ estándar incluyendo la librería "ncurses", resultando una compilación fácil del código.
Para desarrollar este código se implementaron básicamente dos clientes; uno para MiMoD y otro para EyeMoD. Los códigos de estos clientes le otorgan a la UI, las capacidades de E/S necesarias para el fácil acceso a los demonios, es decir: acceso a los datos.
En las diferentes pantallas de la UI, hay vistas globales que muestran el estado de los cuatro detectores de fluorescencia del Observatorio Auger Sur; Los Leones, Coihueco, Los Morados y Loma Amarilla. Por ejemplo, la Fig. 6, muestra un vista del estado de los 24 telescopios del conjunto.

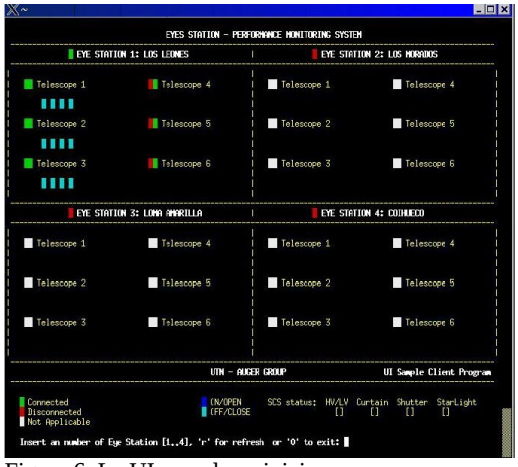

Figura 6. La UI cuando se inicia

Seleccionando un Ojo y telescopio en particular, es posible visualizar un conjunto de parámetros o bien, la información detallada de la matríz de un solo parámetro como se muestra en la Fig. 7.
Aquí, cada parámetro es representado por una matriz de 22x20 variables numéricas correspondientes a los datos en bruto como la mostrada. Alternativamente, se puede optar por visualizar la matriz de resultados de la comparación del mismo parámetro con los valores de referencia.

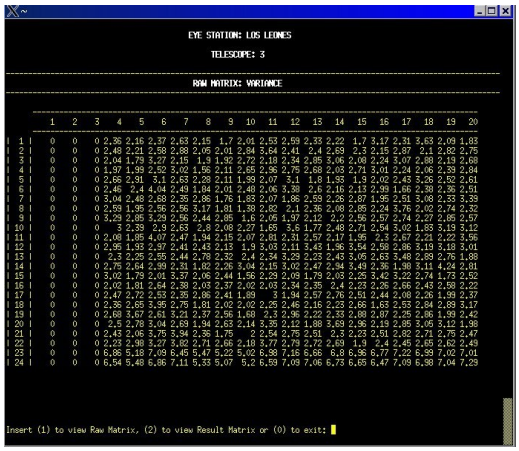

Figura 7. Pantalla de salida de la UI que muestra los datos de un parámetro

5 RESULTADOS

El desarrollo del proyecto descripto se constituyó en un prototipo operativo del sistema de monitor y fue instalado en seis telescopios correspondientes al detector de fluorescencia Los Leones en Malargue y en el sitio de prueba en Karlsruhe.
El sistema funcionó de forma estable en ocasión de las tomas de datos de los detectores durante periodos de al menos 6 horas continuas.
Se configuraron satisfactoriamente todos los ambientes de desarrollo, librerías y compiladores con el marco de trabajo de Auger. La producción de los códigos de programa y su integración con las aplicaciones de monitoreo procedió correctamente.
En esta etapa, algunas porciones de código referido a la adquisición de datos desde el sistema de SCS, no fueron codificados por resultar muy dificultoso el acceso a estos sistemas de control.
Durante las pruebas en campo, se encontraron algunas dificultades cuando interaccionaban los demonios de monitoreo con otros programas de los sistemas en producción en el sitio, especialmente durante las fases de calibración. Estas anormalidades fueron corregidas sucesivamente durante las pruebas pero también demostraron lo poco flexible que es el sistema de adquisición de datos, respecto a los pequeños cambios de configuración por la introducción de nuevos códigos.

6 CONCLUSIONES

La interacción de estos programas de monitoreo junto con los demás componentes del sistema de adquisición de datos debe ser aún evaluada. El funcionamiento en conjunto determinará si su inclusión definitiva será posible o bien se continurá con otros criterios referidos al sistema de monitoreo.

AGRADECIMIENTOS

En el desarrollo del proyecto intervino además la Dra. Beatriz Garcia que fijó los lineamientos generales de este trabajo con la Colaboración Internacional del PAO.

REFERENCIAS

[1] Observatorio Pierre Auger, Información general, http://www.auger.org.ar, [consultada el 20/08/2008]
[2] M. Kleifges, Minutes Auger FD Performance Monitoring Workshop, Forschungszentrum Karlsruhe, Germany, comunicación privada, 2003.
[3] The Technical Design Report (TDR), Pierre Auger Project, draf 2004, Auger Collaboration, http://www.auger.org/admin/, [consultada el 15/08/2008]
[4] CVS, Concurrent Versions System, manuales, http://cvsbook.red-bean.com, http://ximbiot.com/cvs/cvshome [consultadas el 1/07/2008]
[5] El Lenguaje de Programación C++, Edición especial, Bjarne Stroustrup, AT&T Labs, Florham Park, New Jersey, Madrid, 2002.
[6] KDE Development Environment, documentación, http://www.kdevelop.org [consultada el 15/08/2008]
[7] Qt Cross-Platform Application Framework, http://trolltech.com/products/qt [consultada el 15/08/2008]
[8] FD Auger Libreries, repositorio CVS del proyecto Auger, Alemania, ikauger1.fzk.de, comunicación privada, [consultado el 28/02/2006]
[9] Doxigen, Source code documentation generator tool, http://www.stack.nl/~dimitri/ doxygen/, [consultada el 15/08/2008]
[10] CORBA, Common Object Request Broker Architecture, http://www.corba.org/, [consultada el 15/08/2008]
[11] ROOT, Object-Oriented Data Análisis Framework, http://root.cern.ch [consultada el 1/07/2008].
[12] OLE for Process Control (OPC), Final Specification V2.0. OPC Foundation, P.O. Box 140524, Austin, Texas, 1998, www.opcfoundation.org [consultada el 15/08/2008].
[13] Programación en Linux, 2a Edición, Al Descubierto, Kurt Wall et al., Pearson Educación, Madrid, 2001