

**Fermi National Accelerator Laboratory**

**FERMILAB-Conf-99/235-E**

## **Online Monitoring in the Upcoming Fermilab Tevatron Run II**

P. Canal et al.

*Fermi National Accelerator Laboratory  
P.O. Box 500, Batavia, Illinois 60510*

September 1999

Published Proceedings of the *11th IEEE NPSS Real Time Conference*,  
Santa Fe, New Mexico, June 14-18, 1999

## **Disclaimer**

*This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.*

## **Distribution**

*Approved for public release; further dissemination unlimited.*

## **Copyright Notification**

*This manuscript has been authored by Universities Research Association, Inc. under contract No. DE-AC02-76CH03000 with the U.S. Department of Energy. The United States Government and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government Purposes.*

# Online Monitoring in the Upcoming Fermilab Tevatron RunII <sup>1</sup>

P. Canal<sup>1</sup> J. Kowalkowski<sup>1</sup> K. Maeshima<sup>1</sup> J. Yu<sup>1</sup> H. Wenzel<sup>2</sup> J. Snow<sup>3</sup> T. Arisawa<sup>4</sup> K. Ikado<sup>4</sup> M. Shimojima<sup>5</sup> G. Veramendi<sup>6</sup>

<sup>1</sup>Fermi National Accelerator Laboratory, P.O. Box 500, Batavia, Illinois 60510, USA

<sup>2</sup>Institut für Experimentelle Kernphysik, Universität Karlsruhe, Engesserstr. 7, 76128 Karlsruhe, Germany

<sup>3</sup>Langston University, Langston, Oklahoma 73050, USA

<sup>4</sup>Waseda University, Tokyo 169, Japan

<sup>5</sup>University of Tsukuba, Tsukuba, Ibaraki 305-8571, Japan

<sup>6</sup>Ernest Orlando Lawrence Berkeley National Laboratory, Berkeley, California 94720, USA

## Abstract

We describe the online event monitoring systems using ROOT [1] for the CDF and DØ collaborations in the upcoming Fermilab Tevatron runII. The CDF and DØ experiments consist of many detector subsystems and will run in a high rate large bandwidth data transfer environment. In the experiments, it is crucial to monitor the performance of each subsystem and the integrity of the data, in real time with minimal interruption. ROOT is used as the main analysis tool for the monitoring systems and its GUI is used to browse the results via socket, allowing multiple GUI client connections.

## I. INTRODUCTION

The basic designs of the online event monitoring system for both experiments are quite similar. The availability of physics analysis tools, shared memory, a browser, a socket connection, and GUI classes, makes ROOT an attractive choice for online monitoring applications. In both experiments, multiple event monitor programs are attached to the DAQ system, requesting events with desired trigger types. Details of the DAQ systems for both experiments are described in Ref. [2]. ROOT is used as the main analysis tool for the monitor programs. The results from the monitor programs are stored in shared memory in ROOT object format. The main mode of accessing the results is to browse the objects in shared memory with a ROOT GUI via socket connections. In next two sections, we will describe in more detail, the CDF and DØ online event monitoring systems.

## II. CDF ONLINE EVENT MONITORING SYSTEM

The CDF online event monitoring programs are called ‘consumers’. The general definition of a consumer is a process which receives events from the consumer-server [3] in real time. A consumer can also be used for more than monitoring; it could perform other tasks, such as real time calibration. Detailed descriptions of the CDF DAQ system, event builder and Level 3 trigger system, and consumer-server can be found in References [2], [4], and Ref. [3], respectively. The Consumer-server fetches events from the Level 3 system and

passes them to the consumers. Here, we will focus on the CDF consumer framework.

### A. CDF Consumer Framework

A schematic view of the framework with its elements is shown in Fig. 1.

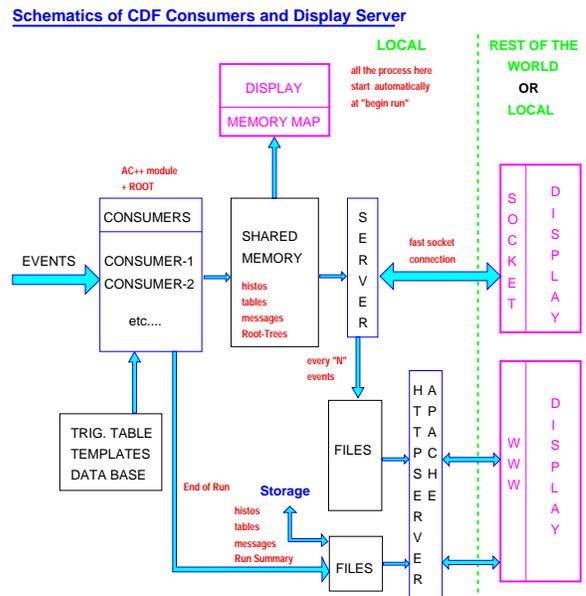


Fig. 1 CDF Online Consumer Framework Architecture

The most important change from the way consumers were run in CDF Run 1 to how they will be run in Run 2 is that the display part and consumer part are separated. The separation of two processes allows us to monitor the experiment locally and remotely with less interruption. The task of the consumer-monitor process is to analyze and monitor the event data and to store the results into shared memory in the form of histograms, tables, and warning messages. These results could then be viewed by the display browser via a server in real time. It is also possible to access the objects in the shared memory directly, however, we plan to use servers as a normal mode of operation. Results of the monitor are also stored as data files periodically during a run, and also at the end of each run which will be archived systematically. These files can be viewed via a WWW server using the same display browser as described

<sup>1</sup>Send questions/comments to maeshima@fnal.gov (CDF), wenzel@fnal.gov (CDF) and Yu@fnal.gov (DO).

above. They should be stored in such a way so that we can readily access them, enabling us to do comparisons with the current data. The display browser provides a GUI to view the online monitored results conveniently, while also providing some basic utilities to do comparisons with previously stored results. The only parts of this framework that need to be run at the experiment are the consumers and servers. By separating the two tasks of monitoring and displaying, we remove the CPU and BUS load associated with displaying graphics from the machine which runs the consumers. Also, network traffic is minimized by serving only small objects (ie. histograms) via the net. During the data taking, multiple consumer processes run in parallel, receiving event data with the desired trigger types from the consumer-server. Different consumers can also run on independent cpu's on different platforms (tested on ilix, linux, OSF). Communication between a consumer and Run\_Control (A process which controls overall CDF DAQ system) is handled using a commercial package called Smart Sockets. The two types of communication that take place here are: 1) The State-Manager in Run\_Control watches the state of consumers. The consumers are started automatically via this process. If for some reason, a consumer dies, Run\_Control will send a message to start the consumer again. 2) Severe errors detected by a consumer-monitor program which are in need of immediate attention are communicated automatically via this connection to the Error\_Handler part of the Run\_Control process.

The consumer framework has three main components:

**Consumers:** In this framework consumers are written in C++ using the CDF offline framework (called AC++) plus ROOT modules which monitor and analyze objects in the event stream. AC++ provides the connection to the CDF offline reconstruction package. ROOT is used for event I/O and its analysis tools. Things typically monitored by consumers are: detector occupancies (dead/hot channels), trigger rate and logic, luminosity, Level 3 reconstruction, physics objects, vertex positions, etc... Specifics of each of the monitor programs are determined and written in collaboration with the experts of each subsystem and are beyond the scope of this paper. The consumer framework provides a template consumer program which includes the basic functions such as input/output, connection to data base (calibrations, trigger-table), and basic classes for the outputs. These base classes allow the server to interpret the stored objects, hence enable us to browse the monitored results efficiently. The template program also provides common utility methods which are useful for monitoring and some examples of how to use them. In the steady state running condition, output format should remain unchanged, however, for debugging purposes we will also provide interactive operations of histogram handling controlled via a text file or a GUI. There are 3 ways to input events to consumers: 1) via consumer-server, 2) read disk files, and 3) generate random events at input stage. 1) is used during the data-taking, 2) and 3) are useful for developing programs and debugging programs/subsystem purposes.

**Display Server:** The display server is a ROOT based program that allows the display browser programs to connect to it as

a client and to access the information in the shared memory. Since it needs access to shared memory it has to run on the same machine as the consumers. Several display/viewer programs can connect from anywhere in the world without having any effect on the consumer itself. The server will handle the requests, giving the process from the data-taking shift crew the highest priority.

**Display Browser:** The display is a ROOT based program that can run on the same or on a different machine with various ways to connect:

- access shared memory directly when running on the same machine. This option should be used just for debugging purposes.
- connect to the Display Server via a socket connection. This is the default way to access the information. The browser should preferably run from a remote machine so that all the CPU-load associated with displaying graphics is transferred to the remote machine. The graphic capability of a simple PC is more than adequate nowadays.
- via the world wide web. There is a plug-in for the Apache web-server which makes the server ROOT aware and allows to access to ROOT files via a web browser like netscape.

## *B. Use of ROOT in the CDF Online Monitoring System*

ROOT has been developed with high energy physics in mind. Not only does it provide the physics analysis tools to replace PAW (what we used on Run 1) but it also provides new features like shared memory, socket connections, web server connections and GUI's which are more sophisticated than what PAW provided. All these features have been programmed using object-oriented design, and they all include examples of how to use their functionality. We use all these features of ROOT in the CDF consumer monitoring system. Although shared memory is implemented for different architectures, we have been working only in the Linux/Unix environment. The classes know about ROOT objects so you can send histograms or ROOT trees with a single command. Creating a data driven client server architecture is relatively easy with ROOT. Other aspects that are interesting for online purposes are the fact that a ROOT daemon is provided which allows one to send root objects over the net. There is also a plug-in for the Apache web-server which makes the server ROOT aware and allows access ROOT files via a web browser like netscape. The CINT[1] interpreter and its debugging capabilities allow fast prototyping to some extent. One needs to be aware that the interpreter has its limitations. However, it is useful and, in principle, not difficult to write programs which can be used as a root macro and can also be compiled to form stand alone programs. We have tested all the components of the CDF consumer server framework and found them to work well. We are currently developing the code that will be used in the final experiment. An example of the CDF online consumer monitor

GUI in development together with some histogram displays is shown in Fig. 2. The right hand histograms are from ADC data taken recently at the CDF experimental hall in Run II format from a portion of the central calorimeters. Left bottom plots are from the Run I data converted to the Run II format. Data were analysed by a consumer template program. The browser displayed the histograms which were stored in shared memory.

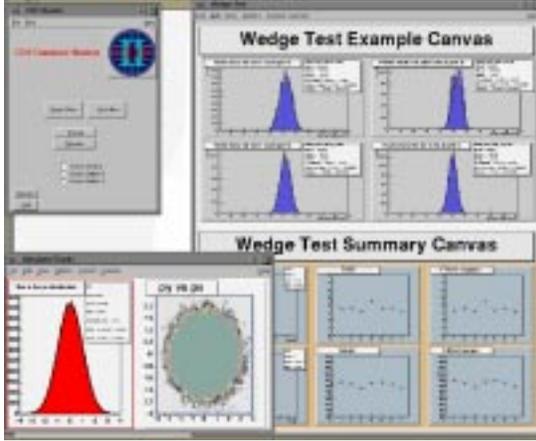


Fig. 2 Example of CDF Online Consumer Monitor GUI in development and some histogram display

### III. DØ ONLINE EVENT MONITORING SYSTEM

The DØ experiment made a decision at the end of the collider Run I in the mid 90s to convert all the existing reconstruction code into C++. This decision affects not only the offline reconstruction but also the entire online system, including communication and controls packages. The DØ online monitoring system consists of three major components:

- Data Acquisition System (DAQ)
- Monitoring Executables (EXAMINE)
- User Interface

These three components can then be further subdivided into smaller components. These smaller components run on one of the three operating systems - Windows NT, Linux, and OSF1 - due to hardware specifications. Therefore the DØ online monitoring system requires portability of software across the operating systems and platforms. The DAQ consists of front-end electronics, two levels of hardware triggers, a software trigger, and data transfer system. Detailed description of the DAQ systems for both experiments can be found in [2]. Figure 3 shows the logical data flow of the DØ online event monitoring system architecture. The system is designed to be fully expandable depending on the bandwidth the system will encounter. A collector/router (C/R) can handle the data being output from multiple level 3 nodes. C/R distributes data to Data Logger (DL) that records the data into files based on the trigger condition a particular event satisfied. This path is tightly controlled to ensure proper normalization and weighting of each event across all the L3 nodes. In other words, if any

of the DL has a problem in writing out an event to a file, all other DL must stop and the deliberate clogging of the data path must be transferred to L3. The C/R also sends all events it sees

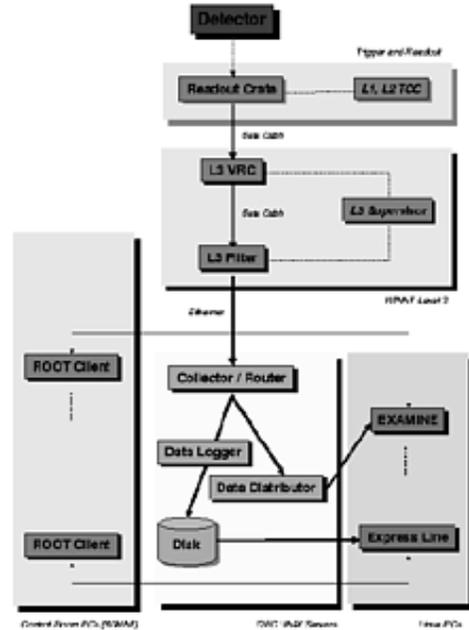


Fig. 3 DØ Run II Online Event Monitoring Architecture

to the Data Distributor (DD) that assigns event buffers and passes the events based on the selection conditions transferred to it by the connected monitoring processes. This path is uncontrolled, because it is desired to continue taking data even if a monitoring process is in a stalled state. It is this path where the event monitoring occurs. Therefore, the entire monitoring system involves, starting from L3, four separate software processes, excluding the GUI, running on three different operating systems and platforms. The executables of the DØ online monitoring run on a farm of PC's operating under Linux while the GUI is most likely to run on a Windows NT.

#### A. The Executable - EXAMINE

The executable portion of the DØ online monitoring system is called an EXAMINE, the name inherited from the monitoring programs of the previous run. There are multiple EXAMINES running on Linux PC servers depending on the purpose the EXAMINE serves. The EXAMINE can be, however, categorized into two large categories. The first is detector performance monitoring which would be mostly geared towards physical quantities that would provide information for detector hardware diagnostics. The second is a global monitoring EXAMINE. This EXAMINE performs full reconstruction of the events to provide information concerning physics objects reconstructed based on algorithms. In other words, this EXAMINE would let the user know how many electrons, muons, or jets have been produced during the run. Therefore, this EXAMINE allows users to obtain an overall quality of the data being taken. This EXAMINE also provides an online real-time event display functionality for

more instructive information in an event-by-event basis. An EXAMINE makes an event selection request via a Run Control Parameter (RCP) file editable by the user. It then transfers the selection conditions - whether by the trigger bit numbers, trigger names, stream numbers or stream names - to the DD, where it makes a connection to the DD via a client-server package, DØME, based on an ACE communication protocol. This request causes the DD to assign an event buffer that is controllable in an RCP file, and at the same time it starts up three separate threads for event transfer communication between the DD and itself. When this process finishes, it starts another buffer whose queue depth is RCP controllable, to store events transferred from the DD to ensure a guaranteed event presence in the buffer for processing independent of the whole analysis process.

The diagnostic histograms are booked and filled in EXAMINE in ROOT format. The EXAMINE also starts up a separate thread for histogram display, interacting with the GUI to allow uninterrupted access of histograms by the user. It also puts histograms in a shared memory for updated accessibility of the histograms while they get filled during the computation. The interaction between EXAMINE and GUI must also allow creation of new histograms, reset of the existing histograms, and deletion of existing histograms while the EXAMINE is running. This would be a bit tricky due to the asynchronous nature of EXAMINE, because the memory might be deleted before the main computational process completes its work, causing conflicts and memory leak. However, we believe we could solve this difficulty.

### B. Use of ROOT in the DØ Online Monitoring System

While ROOT provides a global framework not only for physics analysis tools (PAT) but also for I/O, data packing, Monte Carlo, and GUI, the DØ only uses the PAT and GUI portion of the ROOT, together with minor socket communication. The monitoring executables unpack raw data, reconstruct each event to the level required by individual programs, define and fill necessary histograms, and provide event displays. Figure 4 shows a prototype GUI window built using existing GUI classes provided by ROOT on the IRIX platform. The top portion of the GUI acts as a process registry. While the bottom portion of the GUI acts as histogram control. The main communication protocol to be used for the messaging between the GUI and the EXAMINE executable is CORBA. When a GUI starts up, it does not have any associated processes with it and it first inquire for existing processes to the process registry, a name server to allow, users to attach to a pre-existing process that the user is interested in. This would allow maximally efficient use of computational power and more effective sharing of event statistics. The GUI also provides an ability of starting up a new EXAMINE process on a least used node in the Linux server nodes. The EXAMINE processes are registered to a process registry by the version numbers of governing RCP file. It is this scheme that allows users to parasitically attach to existing processes to access the histogram from shared memory. The histogram control allows various monitoring tools for more effective use of given

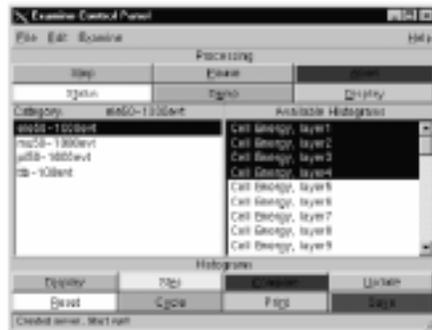


Fig. 4 A Prototype of the DØ EXAMINE GUI

information accessible in histograms. It allows for users to access individual histogram one at a time, viewing a snapshot of the histogram. It also allows the users to select and cycle through one or more histograms continuously, updating the histogram every time it is repeatedly displayed. It allows a continuous updating of the selected set of histograms with a selected update frequency. It would allow comparisons of the given histograms to a reference set, distinguished by the names of the histograms rather than a traditional histogram ID numbers. Complex histogram operations between the current and the reference set are generic functionality of ROOT as a physics analysis tool and the EXAMINE exploits these. The EXAMINE also send out an alarm to the users for any abnormality found from the comparisons.

## IV. SUMMARY

This paper has described CDF and DØ online monitoring systems in the upcoming Fermilab Tevatron RunII using ROOT. We have tested the components of the monitoring systems using test programs and they work well. We are currently developing the full scale software programs to be used in the upcoming RunII experiments.

## V. ACKNOWLEDGMENTS

We would like to thank all the people who are contributing to this work. Special thanks to Rene Brun, Fons Rademakers, Frank Chlebana, Stu Fuess, Dorota Genser, Geri Goeransson, Jerry Guglielmo, Frank Hartmann, Kuni Kondo, Kevin MacFarland, Pasha Murat, Larry Nodulman, Jim Patrick, Tony Vaiciulis, Margaret Votava, DØ and CDF online groups, and Fermilab Computing Division Online and Database Systems department.

## VI. REFERENCES

- [1] Rene Brun, Fons Rademakers, <http://root.cern.ch/Welcome.html>.
- [2] Margaret Votava, *et al*, "Data Acquisition Systems at Fermilab", Abstract 188, IEEE rt99.
- [3] Makoto Shimojima, *et al*, "Consumer-Server/Logger system for the CDF experiment", Abstract 127, IEEE rt99.
- [4] Christoph Paus, *et al*, "Event Builder and Level 3 Trigger at the CDF Experiment", Abstract 104, IEEE rt99.