The DØ Experiment Significant Event System

S. Fuess, S. Ahn, J. F. Bartlett, S. Krzywdzinski, L. Paterno
Fermi National Accelerator Laboratory

L. Rasmussen
State University of New York, Stony Brook

Abstract

A Significant Event System for the DØ Experiment Online Data Acquisition (DAQ) system has been operational since 1992. The system collects and distributes messages related to alarms, heartbeats, and DAQ state transitions. In this paper we give an overview of the hardware and software elements of the system, describe the data flow, give details of the message structure and individual applications, and present an example which illustrates the operation of the system.

I. HARDWARE ELEMENTS

The Online Data Acquisition (DAQ) System for the DØ Experiment was developed with two independent data paths: a high speed customized uni-directional event data path and a standard network bi-directional control and monitoring path [1,2]. A major software component utilizing the monitoring path is the *Significant Event System*, which manages alarm, heartbeat, and run-state transition messages.

There are three principal hardware components of the monitoring path which contribute to the *Significant Event System*: Front End systems, a connecting network, and a Host cluster. The Front End systems, which act as the interface to the detector and other environmental monitoring and control devices, are based upon the Fermilab LINAC control system [3,4,5]. The majority of approximately 35 Front End systems are based upon a 68020 processor residing on a Motorola VME133A card, accompanied by a memory card, a Token Ring interface card, and a utility card which drives an external monitor. Each processor has access to the VME bus of the crate within which it is located, access to other VME crates via a Vertical Interconnect bus extender, or access to other monitoring devices via a MIL-1553B serial link. The remainder of the Front Ends are individual IBM PC systems, which acquire information via various external connections. All of the Front Ends operate by continually repeating (at 15 Hz for the VME based systems) a cycle of data acquisition to fill a local data pool, compare the readings to a local database of analog nominal and tolerance or binary nominal values, and generate and/or process messages. Each Front End system can monitor several thousand analog and binary channels.

The Front End systems are all connected to a Token Ring network. A set of three identical and parallel Gateway nodes act to connect the Token Ring to the Ethernet network used by the remainder of the control and monitoring path elements. Each gateway is a single-board MicroVAX computer running the VAXELN operating system. The peak capacity of each node is approximately 50 to 80 kilobytes per second, depending upon the record size. At peak load, during the trigger condition downloading phase of running, the Token Ring LAN operates at approximately 30% of capacity.

The Host system for the DØ Experiment is a VAX and Alpha mini-computer and workstation cluster running the VMS operating system. In addition to the event data acquisition tools, a suite of applications dedicated to the collection and monitoring of *significant events* runs on this system. These applications will be described in Section 3.

II. DATA FLOW

Figure 1 illustrates the data flow for the *Significant Event System*. The central application in the system is the *Alarm Server*, to which all messages are sent and from which all messages are distributed. There are several message types passed among cooperating applications, the most important of which is the *significant event* message (also referred to as *alarm* message). The other message types are heartbeats, filter profiles, database requests, database replies, and run-state control information. The latter group of message types are specific to certain tasks, whereas the *significant event* message is a more general form which serves multiple purposes.

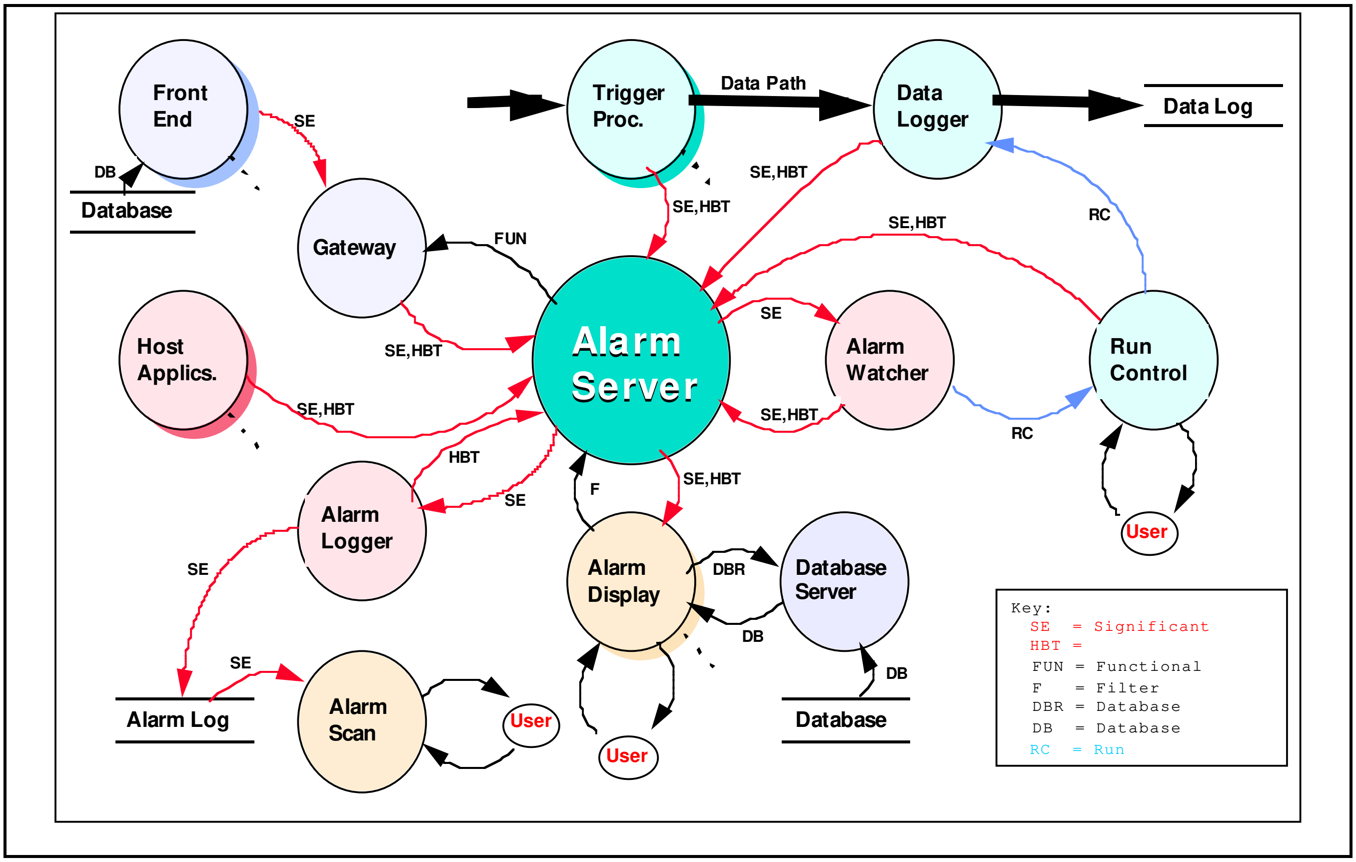

Figure 1: Data Flow of Significant Event System

A *significant event* message contains the following information: a state transition code indicating a good to bad transition, bad to good transition, or informational only; a device and an attribute name; the front end system (or other application) identity and local identifier; a corresponding database identifier; a priority value from 0 (low) to 255 (high); the time and date; and a supplementary block which depends on the nature of the device. For *analog* devices the supplementary block contains the nominal, tolerance, and current readback values. For *binary* devices, the supplementary block contains the nominal bit value and the current readback. There is also a *comment* class of devices (principally used by software applications) which has a 128-character string in the supplementary block.

Figure 1 indicates the flow of *significant event* and other messages. The next section describes the applications that produce and consume these messages.

III. APPLICATIONS

There are three classes of applications within the *Significant Event System*. They are the *significant event* generators, the Gateway, and the *significant event* consumers.

A. *Significant Event Generators*

Significant events are generated by Front End systems, Host applications and other software applications within the Data Acquisition path. The Front End processors, upon noting readings which are inconsistent with their local databases of nominal and tolerance values, place messages on the Token Ring. Such messages are multicast with an identifying group functional code. On the Token Ring the *significant event* messages are framed within the ACNET (Fermilab Accelerator network transport) protocol, which allows for an accompanying data format block indicating the elemental composition (bytes, words, long words, quad words, floating point, and strings) of the message.

A major component of the data path is a set of processors running the software trigger code. These processors are VAX workstations using the VAXELN operating system, and executing FORTRAN and PASCAL reconstruction and filter code. A library of routines callable from these languages within ELN is provided by which *significant event* messages can be generated and transmitted by DECNET to the Host system.

Applications can also generate *significant event* messagesn the VMS Host system. A set of routines for VMS is provided to generate and transmit the messages by mailbox (local) or DECNET (remote). The typical suite of applications includes run control, event logging, event monitoring and detector monitoring tasks.

B. Gateway

The Gateway processors also run the VAXELN operating system. The purpose of these nodes is to provide the interface between the Token Ring and Ethernet physical layers and additionally between the ACNET and DECNET protocols. As previously indicated, ACNET protocol messages have a format block that describes the internal data structure. The Gateway tasks use this information to perform the appropriate conversions to account for the byte order and floating-point representation differences between the Front Ends and the VAXes.

The Gateways maintain independent logical connections to all DECNET clients, including the important connection to the *Alarm Server* task. Each client is allowed to select the addressing modes of Token Ring messages in which it is interested; the *Alarm Server* picks the messages with the group functional code assigned to significant event messages, and hence sees only such messages. The Gateway tasks also buffer incoming and outgoing messages for each remote client, and hence improve the overall bandwidth on each logical circuit.

C. Significant Event Consumers

A set of applications exists on the VMS Host system to process *significant event* messages and to provide information to the detector users. These applications are written principally in PASCAL with some FORTRAN. All are based on a common layered structure, with application specific routines calling routines from a generalized client / server package, which uses an InterTask Communication package, built upon either asynchronous VMS mailbox (local) or DECNET (remote) task-to-task communication.

The client / server package provides a common framework in which the internal message buffering and queuing, error handling, and monitoring actions are provided for the shell application. The basic element of the package is the logical circuit, with utility routines to establish and break network connections and transmit messages. Customized callback routines may be specified to handle any abnormal condition. In the current implementation all activities are queued asynchronously and processed synchronously. In a future implementation each circuit's activities will occur within an independent POSIX thread of the application.

The client / server package includes several features which contribute to the robustness of these applications. The first feature is that of guaranteed message delivery from the server to clients. Any message that cannot be immediately and successfully transmitted is retained and marked for retry. Every five seconds the server will attempt to resend messages; after ten failed attempts the server will disconnect the client process. The disconnection is an indication to the client, once it recovers from whatever caused its halted state, to reconnect and continue its activities.

Another feature of the client / server package is automatic reconnection of clients to servers. In the event of any disconnection of the logical link between client and server, the client will continually attempt to reestablish the connection every 60 seconds.

The *Alarm Server* task is the central point of the *Significant Event System*. It runs continuously as a detached process on one of the Host system processors. All *significant event* and heartbeat messages are directed to the *Alarm Server*. It distributes all new messages to any clients that have requested such. The *Alarm Server* also monitors all heartbeat messages from critical processes and will internally generate a *significant event* indicating the failure of a process should its heartbeat cease. The *Alarm Server* maintains an internal list of all devices currently in a bad state as indicated by a *significant event* message; hence any newly connecting client process can be informed of the complete state of the experiment. In conjunction with the *significant event* message, the *Alarm Server* also stores a record indicating whether a bad condition has been acknowledged; this record can be generated either manually by an *Alarm Display* task or automatically within the *Alarm Server* by the receipt of *a significant event* which is more fundamental. An example of the latter is the 'off' condition of a device, which is more fundamental than *significant events* associated with individual attributes such as voltages and currents being out of tolerance.

The *Alarm Logger* application is a receiver of *significant events* and thus a client of the *Alarm Server*. It writes each *significant event* message as a single record in a sequential file. In order to avoid filling disk files with oscillating devices, the *Alarm Logger* actually delays writing the record for 60 seconds; if a subsequent message arrives in that time with the opposite state transition for the identical device, then the pattern is altered so as to eventually record only the first and last messages of the sequence along with the appropriate counters. An accompanying user task, the *Alarm Scan*, provides an SQL-like interface to the log files; for example a user may specify a time period and a device name to examine its history.

Another receiver of significant events is the *Alarm Watcher* application. This task explicitly requests that only messages above a certain priority level be transmitted. The priority threshold chosen is that associated with *significant events* that affect the quality of data. The *Alarm Watcher* maintains an internal queue of 'bad' messages; upon the transition from zero to greater than zero this task sends a message to the DAQ *Run Control* task to pause further data acquisition. This action is announced to the operators via a DECtalk speaker. Once corrective action has been taken (which should clear any 'bad' condition) then the DAQ operator manually continues the run, also entering log information which is eventually used to construct a downtime report.

The *Alarm Display* task is the principal user interface to the *Significant Event System*. It is also a receiver of *significant events*. Users may request only specific messages by specifying a set of filter condition groups, or may receive all *significant events*. A graphical display, constructed using the MOTIF windowing system, categorizes *significant events* into 'bad', 'acknowledged', and 'good' messages for each filter group and presents summary counter buttons. The user may select any such counter button to get a list of devices that have generated the messages. From this list window the user may further select a single device for numerical and textual information or launch a parameter page control application for the device. To supply the operator with detailed information on the device, the *Alarm Display* uses the database identifier encoded within the *significant event* message as the key to making a database access. The operator may also choose to acknowledge the significant event by entering an identifying comment. A message indicating the acknowledgment is returned to the *Alarm Server*, which generates a new *significant event* propagated to all potential clients.

The main database for the operation of the DØ control and monitoring software utilizes DEC RDB. It was found that applications directly accessing the database suffered in performance when opening the database for use, and also required significant process resources to work effectively. As a result, a *Database Server* application was created from the same set of client / server tools. Tasks accessing the database are linked with a library of client routines that send messages to a server task, the server accesses the database, and the results are returned in a message. The *Database Server* task is given substantial priority and resources, so as to centralize such needs in a single process.

IV. OPERATION

We present here an example to illustrate the operation of the *Significant Event System*. Consider the case where a critical device goes out of tolerance. The Front End monitoring this device will generate an ACNET protocol, high-priority *significant event* message and multicast it on the Token Ring.

A Gateway task will recognize this message as belonging to the group functional code requested by the *Alarm Server* task and enter the message into the input buffer for that circuit. As the input buffer is processed, a data format conversion occurs. The resulting message is entered into an output buffer for DECNET transmission to the *Alarm Server* on the Host VAX cluster.

The *Alarm Server* receives the incoming DECNET message and places it on an input queue. As the message is processed the internal state record of the experiment within the *Alarm Server* is updated. Client processes are checked to see if *significant event* messages of this type have been requested; if so, the *Alarm Server* transmits the message either by local VMS mailbox or remote DECNET connection.

One receiver of the message is the *Alarm Watcher* client, which is selectively monitoring high priority *significant events*. The message is added to its internal queue; if this is the first such message then the *Alarm Watcher* commands the *Run Control* process to interrupt data acquisition. A DECtalk voice indicates the pause of the DAQ system.

The operator, having been alerted by the DECtalk message and the pausing of the run, interrogates the *Alarm Display* for the cause. A new entry has appeared in the category(s) associated with this particular *significant event*; the operator selects the appropriate buttons and list items to determine the nature of the problem. The original fault can be corrected, resulting in a 'bad to good' *significant event* being generated, or the operator may choose to acknowledge the fault and continue running. The original 'bad' *significant event* message is superseded by the new 'good' or 'acknowledged' *significant event*. As the operator resumes the DAQ system, a log entry is made of the fault category.

All of this activity has also been transmitted to another client, the *Alarm Logger*, which has written the messages to a disk file. Users may interrogate the log later to determine the circumstances surrounding the fault.

V. SUMMARY

The *Significant Event System* has been used actively at DØ since 1992. It has proved flexible with the ease that member applications can be created or existing applications have functionality added. The system has also proved to be robust, surviving the vagaries of detector hardware failures, network interruptions and application errors. The set of display and logging utilities has provided the experiment's operators with key tools.

In the future DØ will likely base more of its control and monitoring activity on products shared with the HEP community. Many of our current products will be updated to fit within this more general scheme, while retaining those features we have found particularly useful in the experimental environment.

ACKNOWLEDGMENTS

The authors wish to acknowledge the numerous contributions of the Fermilab Accelerator Division Controls Department for their development and support of the Front End and Token Ring systems. We also thank the members of the DØ collaboration for their feedback and contributions to the *Significant Event System*.

REFERENCES

[1] S. Abachi *et al*., Nucl. Inst. and Meth. **A338** (1994) 185.
[2] A. Ahn *et al*., Nucl. Inst. and Meth. **A352** (1994) 250.
[3] R. Goodwin *et al*., Nucl. Inst. and Meth. **A352** (1994) 189.
[4] R. Goodwin *et al*., Nucl. Inst. and Meth. **A293** (1990) 125.
[5] R. Goodwin *et al*., Nucl. Inst. and Meth. **A247** (1986) 107.