The MIDAS Data Acquisition System

S. Ritt[1] and P.A. Amaudruz[2]

[1]University of Virginia, Charlottesville, VA 22905, USA *and* Paul Scherrer Institute, CH-5232 Villigen PSI, Switzerland

[2]TRIUMF, 4004 Wesbrook Mall, Vancouver, BC, V6T 2A3, Canada

Abstract

A new general purpose data acquisition system called MIDAS (Maximum Integrated Data Acquisition System) has been developed for small and medium scale experiments. It can be used in test systems, where a single PC is connected to CAMAC via a PC-CAMAC interface, and in more complex experiments with several front-end computers and analysis workstations. MIDAS currently runs under Windows NT, UNIX, VMS, VxWorks, and MS-DOS. The front-end part can be ported easily to any operating system that supports TCP/IP sockets. A speed optimized RPC layer has been developed for communication, which supports sustained data rates of 980kB/sec over standard Ethernet. An integrated slow control system contains a fast online database and a history system. Drivers exist for CAMAC, VME, FASTBUS, High Voltage Crates, GPIB and several PC plug-in DAQ boards. A framework is supplied which can be combined with user code for front-end readout on one side and data analysis on the other side. The online data can be presented by PAW as histograms and N-tuples. A run control program has been written as a user interface to the data acquisition and to the slow control system. MIDAS has been successfully used in several experiments at PSI and at TRIUMF.

I. INTRODUCTION

The historical reason for the development of MIDAS has been the upcoming Pion Beta Experiment [1] at PSI. This experiment requires about 2000 channels to be read out at data rates of up to 1MB/sec. Additionally, an elaborate slow control system is necessary to monitor the detector and to keep the operation conditions constant over long periods. The system should utilize modern hardware and operating systems and should be flexible enough to accommodate small experiments and test set-ups. The work was started in 1993 at PSI and was joined by TRIUMF in 1996. The outcome was a general purpose DAQ system that is flexible enough to suit the needs of many different experiments.

II. THE MIDAS LIBRARY

The MIDAS system consists of a library and several applications. Each program participating in an experiment is called a "client". The library is written completely in the C programming language. A two-layer approach is used for easier portability. The lower layer covers all operating system dependent calls like shared memory access and semaphore operations. The upper layer sits on top of the lower layer and is operating system independent. Since the lower layer contains only 7% of the code, porting the library to different operating systems is fairly easy, as long as basic functionality such as shared memory is available.

A. Buffer manager

The MIDAS buffer manager is a collection of functions for event collection and distribution between different clients. A buffer is organized as a ring-buffer located in shared memory. Multiple "producers" can send events into a buffer and "consumers" can receive these events. Each consumer can place one or more event requests in a buffer to allow delivery of all or only specific events. A request can specify to receive all events of a given type or only as much as possible without slowing down the producers.

Events can be distinguished by their event ID and trigger mask that are stored in the event header as can be seen in Table 1.

Table 1
MIDAS event header.

Item	size
Event ID	16 bit
Trigger mask	16 bit
Serial number	32 bit
Time stamp	32 bit
Data size	32 bit
... user data ...	variable

While the structure of the event header is required by the system, the user data area can contain data in any format. Library routines currently exist for the YBOS [2] bank format and a MIDAS bank format.

Since the event buffers are stored in shared memory, locking mechanisms like semaphores must be used to access it. Since these operations are time consuming if performed on an event-by-event basis, a cache memory can be defined between a producer (consumer) and the shared memory. In this case, the whole cache is written to (read from) the shared memory in one operation. The cache is transparent to the clients. Using this scheme, a sustained transfer rate of 15 MB/sec between local clients can be achieved. If the client runs on a remote computer like a front-end, remote procedure calls (RPC) are used for communication. The RPC layer uses similar caching techniques to achieve data rates of 980 kB/sec over standard Ethernet and 4.5 MB over Fast Ethernet.

A common problem in DAQ systems is that a crashed client, like a user analyzer, can block the whole system. To overcome this problem, a watchdog system has been implemented. Every client attached to a buffer periodically signals that it is alive by writing the current time to its client region in the shared memory. Whenever the process crashes or is killed, the other clients see that the crashed client does not update its time any more. In this case all requests from that client are removed from that buffer automatically to release blocked producers.

B. Message system

The message system is based on the buffer manager. A dedicated buffer is used to receive and distribute messages. Clients can send text messages of different types and attach themselves to the message buffer to receive messages. Programs have been written to forward special messages to a text-to-speech system and to a pager system.

Additionally, all messages are logged in a central ASCII file. Together with the ability for operators to produce user messages this file can represent an "electronic run log" of a whole experiment.

C. Online database

In an experiment, configuration data and parameters are usually distributed over several computers. MIDAS uses a different approach. All experiment relevant data is stored in a central database called Online Database (ODB). It contains run parameters, logging channel information, parameters for front-ends and analyzers, slow control values as well as status and performance data. The main advantage of this concept is that all programs participating in an experiment have full access to all relevant data without having to contact different computers.

The ODB is located completely in shared memory for fastest access of up to 50,000 accesses per second locally and 500 accesses per second remotely. It is hierarchically structured, similar to a file system, with directories and subdirectories. The data is stored in key/data pairs, similar to the Windows NT registry. Keys can be dynamically created and deleted during runtime. A client can register a "hot-link" between a local C-structure and a sub-tree in the ODB. Whenever a value in this sub-tree is changed by another program, the C-structure automatically receives an update of the changed data. Additionally, a client can register a callback routine that is executed after a hot-link has been updated. This mechanism is very powerful to control the behavior of front-end programs and analyzers. It can also be used to control hardware. A front-end registers a routine that propagates parameters from the ODB directly to the hardware. A control program may change the hardware simply by writing a new value to the ODB (Figure 1).

Using this scheme, the control program becomes completely de-coupled from the actual hardware. It communicates only with the ODB and needs no knowledge about physical devices or addresses of nodes where front-ends are running. This is similar to the component model. A piece of hardware is an encapsulated component that exports only a set of properties. The whole interaction with the hardware is done via its properties.

Keys in the ODB can be locked for writing when a run is started. An image of the whole database can be written to the logging channels that reflects a snapshot of the current status of an experiment. The whole database or parts of it can be saved and loaded in a simple ASCII format. This feature can be used to load different sets of parameters for different runs.

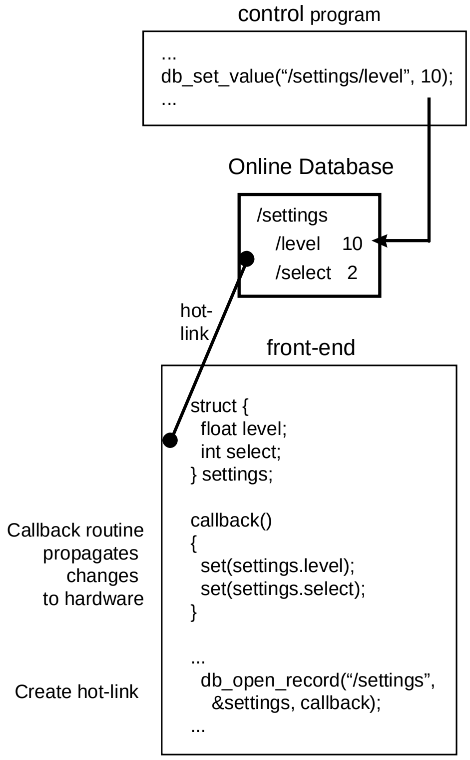

Figure 1: Changes in the ODB get propagated to the hardware by a front-end program

D. Slow control system

The MIDAS slow control system is based on the above described ODB hot-links. It is presently used to control high voltage systems, beam lines, and temperature stabilization systems.

A section with "demand values" and "measured values" is created in the ODB for each device. The demand values are hot-linked to a routine in a slow control front-end that propagates them to the appropriate hardware. When the front-end is idle, it measures the actual values and writes them in the "measured values" section in the ODB. Additionally, events are created which contain the demand and measured values and are injected in the main data stream that is usually logged with the other data on tape.

A program has to perform two simple operations to control a device. It writes new demand values to the ODB and it creates a hot-link to the measured values to receive an update whenever they change. This technique was used in the Pion Beta Experiment for gain calibration. The analyzer evaluated energy histograms to calculate gains for all channels of the calorimeter. Upon user request, the analyzer automatically adjusted the high voltages of all channels in order to equalize the gains.

To access the hardware, a class driver/device driver concept is used. The class driver attaches to the ODB and creates a hot-link to the demand values. It performs periodic measurements of the values and sends an update to the ODB if the values have changed more than a preset threshold. Some class drivers also contain additional functionality like upper or lower limits, ramping with a preset ramping speed and trip recovery for wire chambers. The parameters for these functions are also stored in the ODB. For reading and writing to the actual devices, the class drivers use device drivers. Device drivers encapsulate the device communication. They export only very simple functions like “set value” and “get value”. Device drivers themselves use RS232 or GPIB drivers. This two-level driver concept makes it very easy to add new devices to the system. Only the very basic device driver has to be written while all higher functionality is inherited from the existing class driver.

D. History system

The MIDAS history system stores slow control data and periodic events on disk. It can be queried to produce data-over-time plots. Instead using a standard database, it contains low level file functions to store and retrieve data. It is capable of writing events containing hundreds of values at a rate of several thousand events per second. A query of a value which is written once every minute over a period of one week is performed in two seconds. It is also possible to change the event definition during an experiment.

III. APPLICATIONS

A. Logger

The MIDAS logger supports several logging channels based on different event selection criteria (see II. A.). It can write to disk, to tape, and to remote computers via FTP. The latter feature is used at PSI to write experiment data directly to a hierarchical storage system (Unitree Archive). Data can be written in ASCII, YBOS, or MIDAS format.

The logger can stop a run when a tape or disk is full or when a preset amount of data has been logged. An “auto restart” feature allows logging of several runs of a given size without user attention. At the beginning and end of each run an image of the ODB can be written to all logging channels and to a separate disk file.

B. Front-end and analyzer frameworks

Frameworks have been written which can be combined with user code to form a front-end and an analyzer program.

The front-end framework supports periodic and triggered events. The use of a special polling mode allows for a trigger response time of 8μs (MVME162 under VxWorks) and the possibility of debugging the readout code. Events can be sent to a buffer, to the history system, and to the ODB. User routines can be called on initialization, on run transitions (start/stop/pause/resume), and for the readout of events. The front-end can register itself as a RPC server to execute requested operations like a CAMAC “NAF” server.

The analyzer framework reads events from buffers and calls user routines to analyze them. Based on the event definition that is stored separately in the ODB, the analyzer framework books and fills N-tuples automatically which can be analyzed online with PAW. Other analyzer systems like NOVA and SUSIYBOS have been successfully used together with MIDAS.

C. ODB editor

A simple command line editor has been written to display and change data stored in the ODB. It also contains commands to start and stop runs. Since all data relevant to the experiment is stored in the ODB, the editor can control front-ends, analyzers and the whole slow control system.

D. HV editor

A high voltage editor has been written which runs under Windows NT/Windows 95. It writes new demand values to the ODB upon user request, and it creates a hot-link to the measured values that are displayed in the callback routine. Additionally, it uses an array of channel names for better readability. Channels can be assigned to different groups. Due to the independence of the control part from the front-end part of the slow control system this editor can be used to control other devices like magnets in a beam line without recompilation.

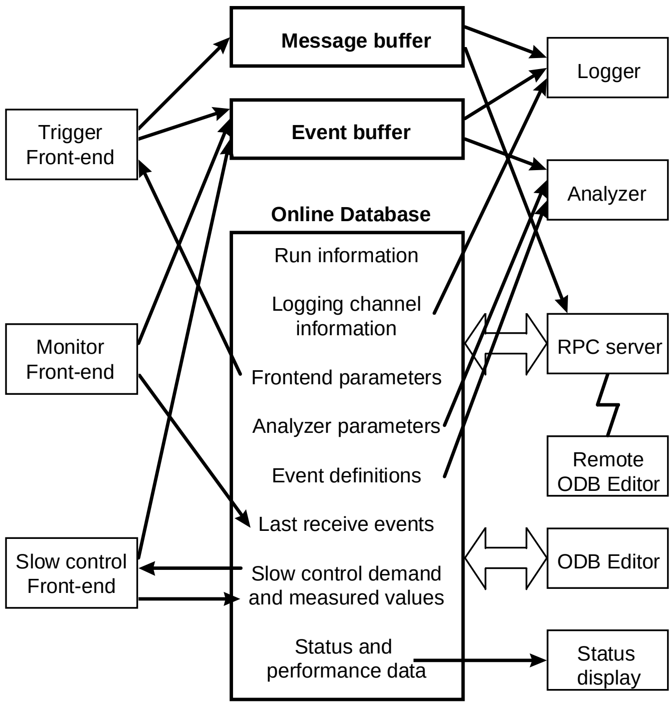

Figure 2: Interaction of different MIDAS components

E. Server

To allow remote access to the ODB and to buffers, a MIDAS server has been written. It is used by front-end computers, remote analyzers and control programs. It can be started interactively or in the background under *inetd* (UNIX) and as a *service* (Windows NT). It creates a sub-process for each connection that protects client connections from each other. A password can be used to restrict remote access. Specific hosts can be excluded from the requirement to supply a password similar to the UNIX *rhosts* file. Due to namespace protection, several independent experiments can run simultaneously on a single machine. This can be useful when off-line analysis from previous runs is performed in parallel to online data taking.

III. Experiments

In 1997, MIDAS has been used in six different experiments at PSI and TRIUMF. The following two chapters describe a small set-up and a larger experiment.

A. Crystal tests at PSI

Several crystal tests have been performed at the πM1 beam line at PSI. A parallel pion beam is directed perpendicular to a scintillating crystal, which is read out by a photomultiplier. The beam position is measured by two multi-wire proportional chambers that are read out by the PCOS III system. One goal of the measurement is determining the spatial uniformity of the crystal.

MIDAS is installed on a PC/Pentium 100 under Windows NT 4.0. The PC is connected to CAMAC via a PC-CAMAC interface [3]. The maximum data transfer rate from CAMAC is 1MB/sec. For small events, event rates of about 5kHz are achieved. The limiting factor in this case is the histogramming of the HBOOK library. To allow direct hardware access under Windows NT, a device driver has been written. The PC runs the front-end, the logger and the analyzer in one process. The analyzer produces N-tuples and histograms in shared memory that can be displayed and analyzed with PAW. The PC is connected to a Exabyte 8mm tape station. Although Windows NT does not allow remote logins like under X Windows, the MIDAS server can be used to grant remote access to the ODB for experiment control. A special version of the *pawserv* program can be used to display histograms on remote PAW sessions.

B. CHAOS at TRIUMF

The CHAOS (Canadian High Acceptance Orbit Spectrometer) detector is composed of about 4500 channels of proportional chambers, drift chambers and scintillation counters. The data acquisition of the different sections of the detector is done via a single VME based processor running under VxWorks. The data are read from several hardware modules of CAMAC, FASTBUS and VME type. The front-end acquisition code runs on an event-by-event basis when a “LAM” is generated by 2 successive trigger levels. A third decision level is made by the front-end software code itself before composing the event. The ODB provides the means of storing and modifying dynamically trigger parameters.

The CHAOS events are pushed from the front-end processor via the buffer manager to the logger process that handles the data storage on two SCSI tape devices in parallel. The CHAOS experiment requires that the data format be in YBOS format. Basic support for the YBOS bank system is part of the MIDAS software package that allows the use of the standard YBOS library for off-line analysis. On-line analysis (SUSIYBOS) is performed by a dedicated computer that samples the data and monitors the experiment.

Pertinent data from several front-end sources are logged separately in the MIDAS history system to allow the user to review data variations over the full period of the experiment. Thirty-two high voltage channels are controlled by a dedicated slow control front-end PC running Linux. The HV editor is running on a PC under Windows 95. A general purpose application called “MIDAS alarm system” monitors the status of the run and experiment conditions as well as any user defined variable of the ODB. This alarm mechanism uses the message system to distribute alarm messages to any MIDAS client. One specific client task running on a PC/W95 processes these alarm messages and forwards them to a text-to-speech converter which “speaks” the messages. Table 2 gives an overview of the computers used in the CHAOS experiment.

Table 2
Computers and tasks in the CHAOS experiment

Hardware	Operating system	Tasks
Motorola MVME162	VxWorks	Front-end event acquisition
DecStation 5000/240	Ultrix	Event distribution, data logging, message logging, history logging
DecAlpha 3000/400	OSF/1	Online event analyzer, experiment control
Pentium PRO 200	Linux	Slow control front-end
PC 486	Windows 95	High voltage control, alarm speaker

IV. Conclusions

MIDAS has proven to be a robust, powerful system for small and medium experiments. The close integration of the event driven DAQ with the slow control system led to a great flexibility, making it very simple to implement all kinds of mechanisms.

The MIDAS source code can be found under http://pibeta.psi.ch/midas. Unfortunately a complete documentation is missing, but several self-explaining examples are supplied to evaluate the system. Further plans include an event builder to combine several event fragments from different front-ends and a graphical control program written in Java.

X. References

[1] http://pibeta.psi.ch

[2] http://www-cdf.fnal.gov/offline/ybos/ybos.html

[3] 1331 IBM CAMAC controller manufactured by Hytec Electronics Ltd., 5 Cradock Road, Reading, RG2 0JT, England