00-150

E10-2000-150

V.G.Olshevsky, S.V.Trusov*

DATA ACQUISITION SOFTWARE FOR **DIRAC** EXPERIMENT

Submitted to «Nuclear Instruments and Methods A»

*Skobeltsin Institute for Nuclear Physics, Moscow, Russia

2000

1 Introduction

The data acquisition (DAQ) software described in this paper was developed for DIRAC experiment[1] which is running now at CERN. The experimental setup consists of forward detectors and two-arm spectrometer. The information comes from about 2000 channels of drift chambers, 2048 channels of micro-strip gas chambers, 480 channels of scintillation fibres detectors and about 150 channels of scintillation and Cherenkov detectors. Pedestal subtraction and zero suppression are implemented at hardware level.

The trigger and read-out electronics schemes are described in the DIRAC proposal[1] and internal notes[2]. It was supposed that one or two accelerator bursts, each of 0.35 – 0.45 sec duration, will be delivered to the experiment beam-line in 14.4 sec supercycle. The main feature of DIRAC DAQ is that all data of subdetectors coming during the accelerator burst are transferred to VME buffer memory modules LeCroy 1190 via FERA bus[3] or stored inside dedicated electronic modules[4] without any software intervention. Relatively slow procedures like checks of data consistency, event building, data transfer, are performed during a pause between bursts and hence the maximal operation rate of DAQ electronics is provided.

The main part of the DAQ software is running on two computers: a VME processor board and a main DAQ host (host computer). The PowerPC based VME processor board (CES RIO 604, 180 MHz, 64 Mbytes of RAM) is running a diskless LynxOS 2.5.1 operating system. This board controls the trigger and read-out electronics via two CAMAC branch drivers (see Fig. 1) and performs event building. The CORBO VME interrupt register[5] is used for generating VME interrupts at the beginning and the end of each burst. These interrupts are handled by a data taking process. We used standard CERN drivers for operating with CAMAC branches and handling CORBO interrupts[6].

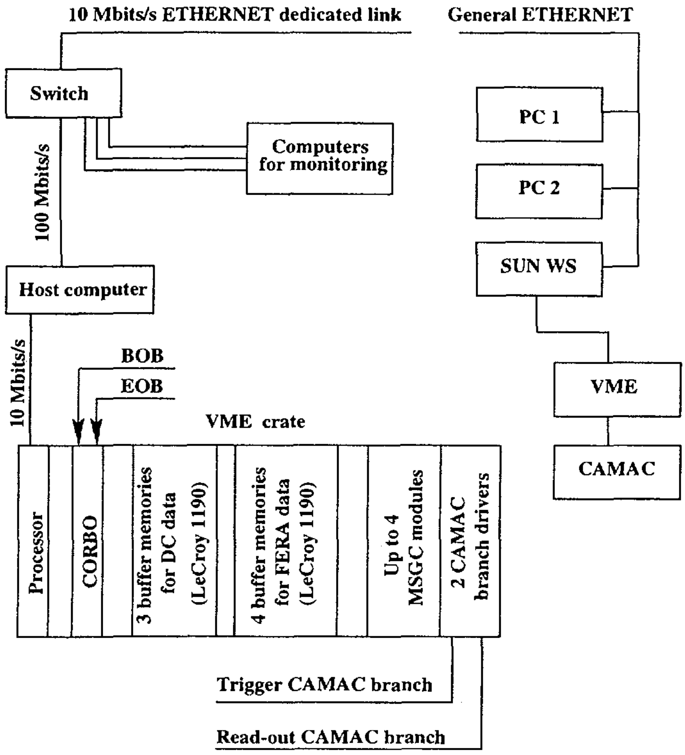

Figure 1: *Hardware layout.* Dedicated Ethernet link is used mainly for the data transfer to a central recorder and to computers (PC1, PC2, SUN workstation) connected to General Ethernet link. VME and CAMAC crates connected to SUN work station contain modules for high voltage control. Ethernet switch decreases traffic on the dedicated link and provides a fast data transfer to monitoring computers connected to the switch directly.

All events accepted in one burst are transferred to the host computer (this one was an HP workstation 715/90, now replaced by a PC running Linux) via a dedicated 10 Mbit Ethernet link and then distributed for recording and for user monitoring programs running on different computers via another network interface.

The DAQ software is written in C under different UNIX-like operating systems (LynxOS, HP-UX, Linux). We used CVS[7] to manage the development of DAQ code.

2 General approaches used in the developing of DAQ software

2.1 Choice of operating system

At the start point of the development of DAQ software it was clear, that DAQ would use computers with different architectures. For example, the VME crate and CAMAC branches were controlled by a Power PC VME-board, the main DAQ host was running on an HP workstation 715/90, and several PCs were dedicated for on-line monitoring.

The VME-board and main DAQ host were running UNIX-like operating systems. Therefore, with many hardware architectures, for providing maximum portability and coexistence of DAQ software we decided to use UNIX-like operating systems on all hosts participating in DAQ and on-line monitoring.

This decision allowed us to use during software development a wide variety of standard UNIX services, including interprocess communication, communication between computers across the network and so on. This also reduced the time necessary for porting the DAQ software to different platforms.

2.2 Guidelines during development of DAQ software

In order to minimize the time necessary for development and maintenance of DAQ software, we used the following approaches as much as possible:

- Whenever possible, we used services provided by the operating system and existing libraries instead of writing own code. Examples: use of RPC calls for slow control processes, shared memory for data distribution, semaphores, inetd services — for data distribution, syslog facilities for providing uniform logging services, XForms libraries for creating GUI, etc.

- Among services, provided by different operating systems, we always preferred those which are most standard for different flavours of UNIX. We also did maximum efforts to write code in a clean and portable manner.

- Whenever possible, we tried to separate software in logically independent pieces (in different libraries, in different programs, etc) in order to make porting, debugging, and maintenance of software less time consuming, and code more robust. Examples: DAQLOG wrapper for uniform access to syslog, SLOWCONTROL libraries, data distribution scheme, etc.

- All DAQ software projects were developed in frames of CVS code management system, which keeps history of changes, makes simpler co-operative code development and process of porting software to different flavours of UNIX.

3 Layout of DAQ software

The schematic layout of the data acquisition processes is shown on a Fig. 2. The DAQ software can be divided into 4 logically (almost) independent groups.

The first is the program for data readout and event building (DT program), which runs on the VME board under LynxOS.

The second group is a set of programs for distributing data across the network for data recording and on-line monitoring programs. This group includes primary data receiver and secondary data senders/receivers. The data distribution scheme was developed in a scalable approach with one data receiver per host. In this group the primary receiver is the only program which receives data directly from the data taking process and is critical for data acquisition — other processes may, or may not be running and may be started at any time.

The third group of programs allows shift members to control the run state via graphical user interface (GUI) — to set run parameters (e.g., run duration), to select trigger type, to change run state (to start, suspend, resume, stop data acquisition). The members of this group are RC process (RC stands for Run Control), which controls run state via slowcontrol calls, RC_gui — the graphical interface for dealing with RC. There is also the RD (Run Display) program, which provides information about current run state and conditions.

The fourth group of programs is a set of slow-control processes, which allow uniform access to hardware on any host from any other host participating in DAQ. The main tasks of these programs are high voltage management, initialisation of CAMAC and VME modules, controlling run state and so on. This group of processes uses RPC calls and provides uni-

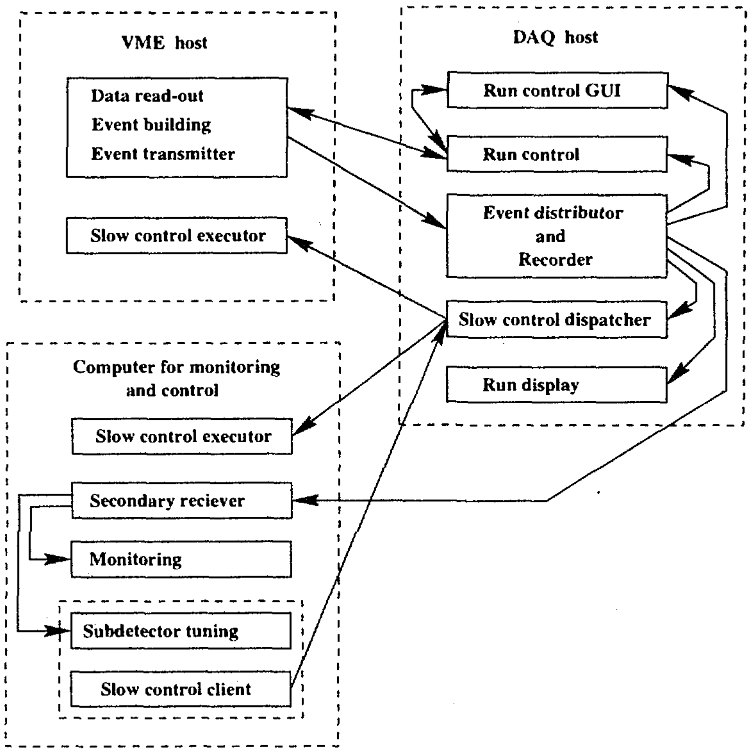

Figure 2: *Layout of data acquisition software.* Basic processes running on VME and DAQ hosts, and on one of monitoring computers. Only main communication lines are shown between different processes.

form access from any host to any piece of hardware controlled by DAQ. It consists of three types of processes: central dispatcher, executors and client programs (clients). Clients perform different operations with hardware via remote calls to central dispatcher. Central dispatcher checks validity of incoming remote calls and forwards them to executors on different hosts. Executors serve remote calls and perform necessary operations with hardware.

Besides these groups of programs, there are several supplementary programs, developed mainly as monitoring and debugging tools.

Software for trigger support and on-line monitoring was developed by other persons and is not discussed in this paper.

All programs mentioned above use a uniform logging mechanism for reporting problems. It is implemented as a library of wrapper functions for standard UNIX syslog facilities. All DAQ hosts are configured so that messages from DAQ processes go to the logfiles on the main host. This helps in a great extent to keep track of current software state and to localize possible problems with DAQ processes. Note that these logfiles contain mainly reports about different events in DAQ software.

Besides the logging features described above there is also another way for storing messages, designed for "physical" information — so called run logbook. This one keeps information which is of interest mainly for data analysis — target type, beam intensity, parameters of read-out electronics and so on.

Now we will consider the programs mentioned above with more details.

4 Data read-out and event building

Data taking process (DT) runs on VME processor board and performs read-out of data, event building and data transfer to main DAQ host.

At start DT initializes few electronics modules used to control data acquisition (CORBO interrupt register and control registers), establishes connection with the primary data receiver and waits until run control (RC) process establishes connection with it. Then DT waits for any interrupt to continue the work. This may be either real-time interrupt from accelerator (begin of burst or end of burst), or timer signal generated after expiring of predefined time out. The installed timer handler forces DT to transfer run status even in absence of accelerator signals in order to inform other processes (including RC) that data taking is in operation. Each time an interrupt signal is obtained and served, the DT checks availability of RC command. Such command, if present, forces DT to take corresponding actions and to changes the state of data acquisition (or run) correspondingly.

The possible states of run are: STOPPED, RUNNING, PAUSED and SC_PAUSED (paused by slow control). In the RUNNING state the run status is transferred together with built events. In STOPPED, PAUSED and SC_PAUSED states the timer signal causes periodical transfer of the run status for informing other DAQ processes that DT is in operation. If a predefined time-out expired and run status fragment was not transferred it means that either DT terminated abnormally or there is a problem with data distribution. To indicate this situation we use UNKNOWN run state.

DT can change the run state when receiving a corresponding command from the run control process. Actions undertaken after receiving STOP, PAUSE and SC_PAUSE commands are evident from their names: data acquisition is disabled, run state is changed, and DT waits until next command is available. The CONTINUE command, which can be issued only when run is in PAUSED or SC_PAUSED state, causes DT to change run state to RUNNING and to wait either for data from next burst, or for next run control command.

NEWRUN command causes the start of new run. As it will be described in Run Control section, RC sends this command to DT after performing initialisation of read-out and trigger electronics and preparing configuration files with predefined names. When DT receives this command, it reads the configuration and parameter files containing the description of subdetectors and read-out electronics, fills internal structures which will be used in the event building, and finally changes the run state to "RUNNING".

Let us discuss in more detail what happens when the state is "RUNNING".

The data may be received only in the interval between begin of burst (BOB) and end of burst (EOB) signals, which are provided by the accelerator. These signals are put into CORBO register and cause real time interrupts. When DT detects the BOB interrupt, it enables triggers and opens read out electronics. During the burst (recall, its duration is 0.35 – 0.45 sec) all signals necessary for accepting data are provided by hardware without any intervention of the program. At the end of the spill, the real time interrupt corresponding to EOB signal informs DT that all data are digitized and put in VME buffer memories.

When receiving an EOB interrupt, DT checks that data from previous burst were already processed and sent to the host computer. If it is so, the program copies data from VME buffer memories to internal buffers and immediately prepares electronics for accepting next burst. If the data of previous burst were not transmitted yet, then data of new burst remain in VME memories until DT will complete transmission of previous one (of course, accepting of next bursts is disabled at this time). Due to this behaviour, the DT program is able to accept very close bursts — minimal interval between first and second bursts is about 0.5 sec with about 1 MByte of data in each.

During event building DT checks data consistency. The read-out electronics of subdetectors provide headers and serial numbers of subevents written in each VME buffer memories. These serial numbers allow one to check that all buffer memories contain the same number of subevents and that different subevents correspond to each other. In the case of an overflow detected in one of FERA or DC buffer memories DT ignores all subevents with numbers starting from that one in which overflow happened. It is not so in a case of MSGC memory overflow: following events will be built without MSGC data (which are absent). It was done to provide a possibility to collect as much as possible data from other subdetectors for pedestal measurements and due to the fact that in off-line analysis it is not required that all MSGCs have data for each event.

The data check includes procedures which are specific to each branch (DC, MSGC or FERA ones). All these checks are based only on the expected structure of data provided by electronics and do not touch the physics characteristics of accepted information. The burst data are ignored when a fatal error is detected in DC or FERA memories, because it is not always possible to recover valid events and an illegal recovery may create problems for off-line analysis. The error detected in one of MSGC memories is not fatal and is treated as the absence of events in it at all.

During the data check the structure of pointers to subevents is filled up. If no fatal error was detected, this structure is used to build events and to copy them into the output buffer containing all built events of the burst. The pointers to events in this buffer as well as a trigger type of each event are stored in another buffer. The DIRAC event format is described in [8]. The data accepted in one burst can be considered as one "record" consisting of events of variable lengths. Each event consists of data blocks which in turn contain event description and data of subdetectors. Subdetector data blocks are filled by data obtained from electronics in the same format as they were read, at least we try to do that for most subdetectors. It was done in order to minimize the number of operations on raw data during event building.

There is a dedicated kind of events which is used for including information from another parts of DAQ software to the data flow. DT process reads such events from the shared memory region, to which other DAQ processes can write with the help of special slow control calls. During event building, DT checks the availability of data in this memory region and builds a corresponding event when necessary. For example, a run may be in PHYSICS or TUNING mode. If TUNING mode was selected than slow control routine can pause the run, change the electronics parameters, continue the run with new parameters. A corresponding message may be written to the shared memory and it allows one to trace what was changed and how it affects on the collected data. This feature was implemented for supporting automatic tuning of the setup.

In order to simplify the control and monitoring of data acquisition status and on-line event monitoring, the dedicated run status fragment of data is filled up for each burst. It contains the run number, current run state, number of events collected in the burst and in the run, running time and some other information. Different DAQ processes may monitor the status of setup and validity of data and set warning or alarm bits in corresponding words. Alarms and warnings can be triggered either at software level, or by providing a hardware signal to a dedicated VME input register. The DT process reads and includes alarm and warning bits into run status fragment. The presence of unmasked alarm forbids data writing to the storage, but the data are still distributed to monitoring programs to give them a chance to analyze and find the reason of the problem. Initial configuration for warning and alarm masks is set by Run Control Graphical User Interface (RC_gui) at the start of run.

The data collected by the data taking process are transfered at each burst to the host computer via dedicated Ethernet link. Its further distribution is discussed in the next section.

5 Data distribution

Processes of this group are responsible for storing data on the disk and for its distribution to the hosts willing to perform on-line analysis. The group includes primary data receiver and secondary data senders/receivers. The data distribution scheme was developed in a scalable approach with one data receiver per host.

The primary receiver runs on main DAQ host. It receives data from DT program and stores them on the local disk for further transfer to central tape recorder with SHIFT utilities[9]. The receiver also puts data into shared memory buffers for further access by other DAQ processes and user monitoring programs. The secondary senders/receivers distribute data to other hosts willing to receive data on-line.

All these programs know nothing about the internal data structure -- their only task is to distribute incoming data across the network. This helps to make code for these programs relatively simple and more stable (because it is not affected by changes/additions in data format).

Both the VME processor board and main DAQ host are put in local network, for which the main DAQ host serves as a router. This is done in order to provide maximum and guaranteed bandwidth for transferring data from VME to host computer.

As soon as data from one burst are read and prepared by data taking process, they are sent to the main DAQ host. All data consist of 32-bit words with a predefined byte order (network byte order) to avoid problems with byte ordering on different computers.

Data transfered from VME are divided in several (to the moment, five) fragments. The first one contains current run status, which was described earlier.

The second fragment is filled with offsets to events and their trigger types, and is dedicated for use by monitoring programs for fast look-up of events.

Events themselves are transfered in the third fragment.

The fourth fragment contains special run header event with description of read-out electronics configuration, target type, magnetic field value, proton flux intensity, and so on. This fragment is sent from VME only once at start of new run, for all subsequent bursts it has zero length.

As we mentioned above, other processes can also prepare special kinds of events. Such events, when available, are transferred in the fifth fragment of data.

The whole data buffer sent from VME contains timestamp, lengths of named above data fragments and data fragments themselves.

5.1 Primary receiver

As we described earlier, the data taking program which runs on VME processor, reads data from hardware and sends them to main DAQ host. The data sent from VME processor are read by the primary receiver program running on the host computer. The primary receiver has two main tasks — to store incoming data into the local disk file, and to put data of the last burst into shared memory region (data pool). Let us discuss both with more detail.

When data arrive (if write to disk file was requested), the primary receiver extracts the fragment containing events and stores it into disk file. It permanently keeps track of the size of current datafile in order to avoid creating enormously large files. When the size of current datafile exceeds the predefined maximum (which is set as an argument at start of the program), the receiver automatically closes the current file and opens the next one. The receiver also closes automatically the current datafile when the run goes to "stopped" state or in case of lost connection with data taking process (the latter may happen at abnormal termination of data taking process). At close, the receiver marks the file as readonly, thus signalling that writing of that file is completed and it can be transferred to the central tape recorder.

Besides storing data in file, the receiver provides data of current burst for consumer processes by placing them into the data pool implemented in shared memory. The primary receiver is the only process which has rights to modify data in the pool, all other processes can only read data.

The receiver sets a semaphore before refreshing information in the pool and releases it after modification ends, thus providing means of synchronisation for processes willing to read data from the pool.

The receiver places data in the pool in the same "fragmented" structure as they were received from the data taking process, but with some modifications — it places most recent run header (fragment 4) and special events (fragment 5) in every burst during the whole run. This is done so because 4 and 5 fragments contain information critical for on-line monitoring programs, and this information should be available at any time.

5.2 Data consumers. Secondary senders/receivers

Since data corresponding to the current burst are placed into the shared data pool, any process can map this pool into its address space and process data on-line. As it was said above, arrival of new data is signalled with a semaphore. This way of receiving data is used by all data consumers, including RD (run display) and other on-line monitoring programs.

Among data consumers, there is a special class of processes which retransmits data to other hosts, we call these processes secondary, or slave, senders. The counterpart of slave sender is slave receiver — the program which runs on host willing to receive data on-line.

If some computer needs to receive data, it starts the slave receiver program. This program connects to a special port of a server host (this may be the main DAQ host as well as any other host which already receives data). When connection with server host is established, standard UNIX inetd daemon launches slave sender program, which in turn reads data from pool for each burst and sends them to the slave receiver. The slave receiver works with shared memory pool in the same way as the primary receiver does — it receives data, sets semaphore, puts data into data pool, and releases semaphore, thus signalling to data consumers on this host that fresh data arrived. Among data consumers on such "slave" host, the next level slave sender may be started for propagating data to other hosts.

As it can be seen from the discussion above, with such approach each host receives the full volume of data only once per burst and can run any number of monitoring programs. It also can serve as data source for other hosts.

As we saw during DIRAC beam-time, the described scheme of data distribution worked in a stable way and was able to distribute up to 3 megabytes of data per super-cycle (in two bursts) to 6 hosts participating in data acquisition and on-line monitoring. The limitation for transfer rate with such scheme arises from Ethernet links used and is about 4 MBytes per super-cycle. This limit can not be reached with the current volume of VME buffer memories. To increase further the transfer rates (for example, in case of increasing volume of VME buffer memories) the network links must be rearranged.

6 Run control

Both run control (RC) and its graphical user interface (RC_gui) are running on the main DAQ host.

The RC process provides a control of the data acquisition — it transmits commands to the data taking process (DT) via special connection. Such commands may be sent by other processes like RC_gui and slow control or issued by RC itself. Only one instance of RC may be running at any given time, it's checked at the start of RC process.

When running, RC reads run status from data pool at each refreshing of data. If a predefined time-out (about 60 sec) for getting status expires, RC considers this as a fatal error in the data transfer or in DT and exits. We should mention that RC may be killed and started again even during data acquisition and it will not affect run state and DT operation. For example, if run was in RUNNING state, the DT continues to get data and to transfer them to primary receiver even in absence of RC process. When restarted, the new instance of RC will establish all necessary connections with DT and provide further control of data acquisition.

The SC_PAUSE command may be issued only by a slow control (SC) function. Typically, any such call requires to pause data acquisition, perform an action, and continue the data acquisition. Execution of SC function may depend on the initial state of run, some procedures may be invoked in PAUSE state and it is natural to restore the state at the end of execution. So, we forced to distinguish PAUSED state caused by RC GUI from SC_PAUSED one in order to escape conflicts when during execution of SC procedure in paused state the RC_gui tries to resume the data acquisition.

For the moment there are two cases when RC itself issues commands to DT. The first case is when the limit of run duration or number of collected events is reached. In this case RC sends STOP command to DT. The other case is when periodical automatic calibrations were requested at start of new run. The required numbers of bursts between different modes of measurements are transfered in the run status fragment. The RC checks if is it time to perform a calibration, and if it is so, it sends SC_PAUSE command to DT, waits until this command will be completed and after that reloads read-out and/or trigger electronics via calls to SC procedures. As electronics is reloaded, the RC issues CONTINUE command and waits until several bursts with changed parameters are accepted. After collecting calibration data, RC repeats the same steps in order to restore standard run parameters.

The run control graphical user interface is written on the basis of XForms library[10] and works in X11 environment. It provides means for manual control of run by accepting user commands and transmitting them to RC process.

When started, it checks that there is no other RC_gui process. As RC_gui transfers commands to the RC process, the last one must be running as well. The information about current run state is obtained by RC_gui from the data pool.

RC_gui opens the primary control panel, waits until the run status fragment is obtained and sets internal variables in correspondence with it. Only after that control buttons are enabled and may be used to perform an action.

As it was mentioned above, performing commands by DT is synchronized with VME real time interrupts or time-out signals. So, a reaction time of DAQ on the issued command may take one or more sypercycles. RC_gui may be killed and started again without affecting on other DAQ processes.

At the start of new run RC_gui opens a series of windows which allow the user to set required run duration and trigger type, to decide how to serve warnings and alarms. It also provides possibility to control read-out electronics modules — to switch modules on or off or to change their parameters. This can be done for any subdetector in total as well as for individual modules.

When all these parameters are set by the user, they are saved to several configuration files with predefined names and other processes can access them. DT includes the essential part of this information into run status fragment and into dedicated run header event, so it can be used during on-line and off-line data analysis.

When the user completes the selection of new run parameters, RC_gui transmits NEWRUN command to RC, which in turn calls SC functions to initialize the electronics and only after that retransmits the NEWRUN command to DT.

All other commands are retransmitted to DT via RC without any additional operations.

In order to monitor the data acquisition process a run display (RD) program was developed. It is also built based on XForms library for X11 environment. It may run on any host receiving data on-line and displays some run parameters, current run status, amount of collected events in the burst and in run, running time, the volume of data in the burst, and so on. It shows also general information about warning and alarm messages, and the status of MSGC, DC and FERA VME buffer memories.

7 Slow Control

The purpose of this group of processes is to perform relatively rare and slow operations like CAMAC modules initialisation, high voltage control and monitoring, and so on.

The hardware used in DIRAC experiment is controlled by different computers. In order to provide uniform access to all controlled hardware, we implemented slow control as a distributed system with central dispatcher which serves (forwards) all calls from client applications to hosts which perform real actions with hardware. Such approach allows one to keep track of the state of the whole system at any moment. Moreover, such approach made it possible to implement behaviour where certain procedure can be called only in "compatible" system states. For example, procedures which change hardware parameters are not allowed when state is "RUNNING" and central dispatcher rejects such calls.

The slow control scheme uses RPC (remote procedure calls), which are standard part of any UNIX-like OS. Therefore, this set of programs is fairly portable.

Each host which owns a piece of controlled hardware, should have the server part (slow control executor) running. Such executor contains some set of procedures specific for hardware controlled by this host. Client application which may run on any other host, can call these remote procedures. All requests from client applications do not go directly to executor programs but instead call some procedures of a central dispatcher. The dispatcher process serves several purposes:

- It allows one to group several remote procedures in one logical call, thus making cleaner the code of client applications. For example, at hardware initialisation several remote procedures should be called — initialisation of readout modules, loading of parameters, loading of trigger logic, etc. All these calls to remote procedures are encapsulated in one routine of central dispatcher, so client applications should call only it.

- It hides "implementation details" of remote procedures from client applications. This helps to minimize efforts in case of changing some routine, because such modifications should be done only in one place — in central dispatcher. For example, if it is necessary to add something to initialisation routine, this should be done only in the code of the central dispatcher, and client applications which use it would remain unchanged.

- It keeps track of current state of all DAQ system. This is useful because some procedures can affect run conditions and should not be used in certain run states. Having the information about current run state, dispatcher addresses this problem by selectively rejecting calls from client applications to such "dangerous" procedures.

- It can communicate with RC (run control) process and can suspend and resume run. For this purpose, special run state SC_PAUSE (pause from slow control) is reserved. This functionality is implemented for supporting applications for automatic tuning of setup. The idea is to allow application program to suspend run, change some setup parameters (recall, such changes in "RUNNING" state are rejected by dispatcher), and after that resume the run and collect data with new conditions. We should note that dispatcher allows such behaviour only when run is in "TUNING" mode. When run mode is "PHYSICS", all changes of run conditions are disallowed.

8 Conclusion

Programs described above worked several months during DIRAC runs at the CERN PS accelerator. As these runs showed, the DAQ software works in a stable way, is reliable and easy to manage.

The whole system was able to accept data from two bursts in an accelerator supercycle with up to 1.5 MBytes in each, and to distribute events to hosts participating in on-line data processing. We should note that this limitation comes not from DAQ software, but arises from the volume of VME buffer memories currently used.

9 Acknowledgements

The following people have contributed and are thanked for their help. V. Karpukhin and A. Kulikov developed and tuned the general trigger and read-out electronics scheme. I. Manuilov participated in a lot of discussions concerning the global scheme of DAQ processes and developed a user library for event access as well as event monitoring code. V. Iazkov for help in understanding off-line needs. L. Afanasev and M. Gallas for fruitful collaboration. M. Ferro-Luzzi for the support of work on the initial stage. D. Drijard and L. Nemenov for fruitful discussions and permanent support of the work. S. Soulie and G. Polivka for successful solving of system manager problems. C. Detraz for technical support. Our special thanks to C. Moine, J. Neiwold, J. Petersen, M. Joos and I. Perrin for very useful consultations and help in our work. And, of course, to all our DIRAC collaborators for their understanding of the difficulties in the development of DAQ software in a rather limited time and for contributions in tests and debugging of the system as whole.

References

[1] Proposal to the SPSLC, CERN/SPSLC 95-1, SPSLC/P 284, Geneva, 1994 (updated 1995): Lifetime measurement of $\pi^+\pi^-$ atoms to test low

energy QCD predictions, DIRAC collaboration (B. Adeva et al.)

[2] NOTE 96-26 : First level trigger for DIRAC, A. Kulikov (JINR)

NOTE 96-16 : A comment on the level-2 trigger, K. Kuroda (CERN)

NOTE 96-17 : Note on trigger logics, F. Takeutchi (FAROS)

NOTE 96-27 : Third level trigger for DIRAC (versions of implementation), V. Karpukhin, A. Kulikov (JINR) and V.Yazkov (SINP MSU)

NOTE 96-31 : A note on level-2 trigger logics, M. Kobayashi (KEK) and K. Kuroda (CERN)

NOTE 97-01 : Ionization hodoscope: a trigger scheme, C. Bricman, J. Buytart, C. Detraz and M. Ferro-Luzzi (CERN)

NOTE 98-01 : Fast zero level trigger. A. Kulikov (JINR)

NOTE 98-02 : Trigger scheme of the ionization hodoscope. A. Kulikov (JINR)

NOTE 98-07 : The complete software-programmable third level trigger for DIRAC. M.V. Gallas (University of Santiago de Compostela, Spain)

[3] LeCroy Research Systems Catalog, p.108–115, 1999

[4] NOTE 2000-01: MSGC/GEM Detector electronics. F. Gomez, P. Vazquez, (University of Santiago de Compostela, Spain).

[5] RCB 8-47 CORBO VME Read-Out Control Board, User's Manual, 1993, CES

[6] Software support, CERN,
http://www.cern.ch/ess/os/services/software_support.html

General Purpose Interrupt Handler for the CES RIO2 8060 and the RTPC 8067 running LynxOS, M. Joos, J. Peteresen, 1996, CERN,
http://atddoc.cern.ch/Atlas/Notes/019/Note019-1.html

CAMAC routines for OS-9/VMEbus CBD8210 Systems. Jorgen Petersen, Per Scharff-Hansen, 1992, CERN,
http://wwwinfo.cern.ch/ce/ms/os9/papers/CAMAC_OS-9.html

[7] Version Management with CVS, 1992, 1993 Signum Support AB. Per Cedegvist et al.

[8] The DIRAC Offline User Guide, D. Drijard (CERN), M. Hansroul (CERN) and V. Yazkov (SINP MSU), 1999, CERN http://www.cern.ch/DIRAC/Userguide.html#I/O

[9] SHIFT — the Data-Intensitive Physics Analysis Facility, CERN, http://wwwinfo.cern.ch/pdp/serv/shift.html

[10] Forms Library, A Graphical User Interface Toolkit for X. T.C. Zhao and M. Overmars, 1996,1997 ftp://einstein.phys.uwm.edu/pub/xforms

Received by Publishing Department on June 28, 2000.