EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

CERN - ECP / 95-12
24 May 95

Review and Prospects of the CASCADE Data Acquisition System at CERN

D. Burckhart, C. Bizeau, W. Bozzoli, R. Divia`, F.Lamouche,
L.Gavino, J.-P. Matheys, Y. Perrin, J.Petersen, A. Saravia,
L.Tremblet, P. Vande Vyvre, A. Vascotto, P. Werner

ECP Division, CERN, 1211 Geneva 23, Switzerland

Abstract

CASCADE, a multi-processor real-time data-acquisition system for HEP experiments developed at CERN by the ECP-DS group, has now been in operation for one year. The current implementation supports configurations based on VMEbus processors running OS-9 and on UNIX workstations interconnected via VICbus or Ethernet. The project is reviewed by describing the main characteristics of the package, the applications in which it has been used, and the results of this experience. The main improvements of 1994, which include a parameterized multi-level event builder, a remote monitoring option and a powerful run control facility, as well as ongoing developments and prospects for 1995, are presented.

Presented at the Ninth Conference on Real-TIme Applications of Computers in Nuclear, Particle and Plasma Physics, RT-95, East Lansing, Michigan, USA, May 22-25 1995

Review and Prospects of the CASCADE Data Acquisition System at CERN

D. Burckhart, C. Bizeau, W. Bozzoli, R. Divia`, F. Lamouche, J. L. Gavino, J.-P. Matheys,
Y. Perrin, J.Petersen, A. Saravia, L.Tremblet, P. Vande Vyvre, A. Vascotto, P. Werner
ECP Division, CERN, 1211 Geneva 23, Switzerland

Abstract

CASCADE, a multi-processor real-time data-acquisition system for HEP experiments developed at CERN by the ECP-DS group, has now been in operation for one year. The current implementation supports configurations based on VMEbus processors running OS-9 and on UNIX workstations interconnected via VICbus or Ethernet. The project is reviewed by describing the main characteristics of the package, the applications in which it has been used, and the results of this experience. The main improvements of 1994, which include a parameterized multi-level event builder, a remote monitoring option and a powerful run control facility, as well as ongoing developments and prospects for 1995, are presented.

I INTRODUCTION

CASCADE [1] [2] is a distributed, multiple-platform real-time data-acquisition system developed at CERN by the Data-acquisition System group of the Electronics and Computing for Physics division. CASCADE provides services for data collection and buffering, data flow across the data-acquisition chain, event building and event sampling for on-line monitoring, data recording and run control. Originally developed following a request from the NOMAD [3] experiment, CASCADE has been designed to adapt to a wide range of applications and system configurations. System extension and reconfiguration often imposed by changes in the setup of experiments, require flexibility in the data-acquisition design. A high degree of modularization of its components facilitates the integration of the most recent advances in technology.

To accommodate these design demands, the CASCADE system is composed of a small number of building blocks and of their connection units.

Fig 1. CASCADE Unit construction system and flow of an event through a stage

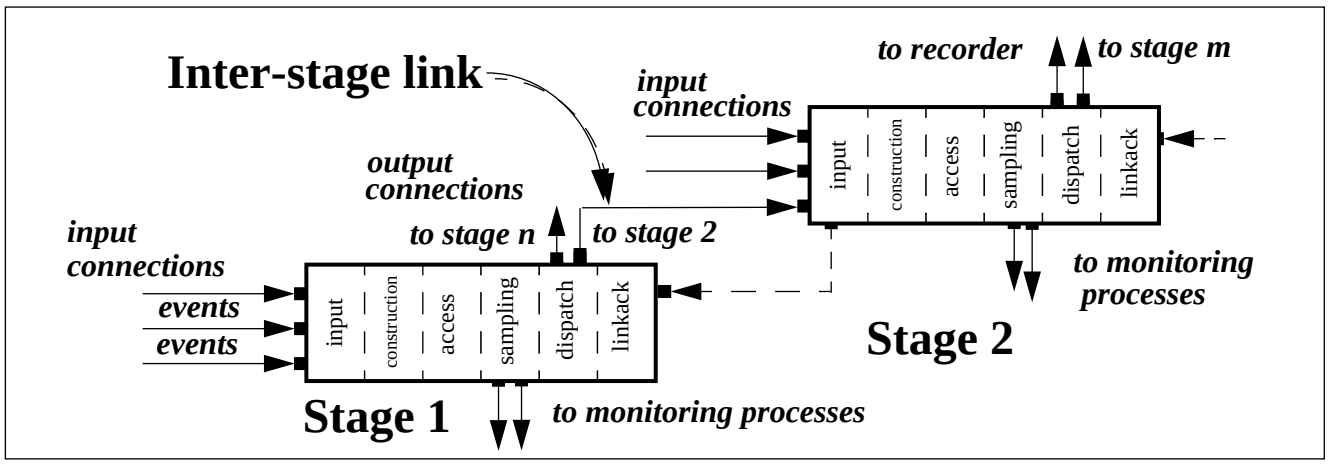
II THE UNIT CONSTRUCTION SYSTEM CONCEPT

The CASCADE system has been designed with the aim of achieving a high degree of homogeneity, flexibility and scalability. Therefore the concept of a unit construction system has been followed. The basic software unit, called the *stage*, and the stage connection unit, called the *inter-stage link*, have been defined as illustrated in Fig. 1. Plugs for special-purpose connections like event monitoring, run control and experiment-specific functions are provided. This approach allows for easy task distribution and their communications in a distributed system using a variety of operating systems and physical hardware connections. Small and large configurations can be built from these two components.

The stage operates on event data at a given level of processing: it controls the data transmission through its input and output ports, if necessary it merges event components into full events, and it provides data access to processing and monitoring programs. Equally, event recording is performed by a specialized stage. The run control process and event-monitoring programs access the stage via special purpose connections. Application-specific functions, connected to the stage, allow for experiment-specific tailoring. The inter-stage communication is based on a set of interfaces and protocols which hide the details of the transfer mode.

The physical architecture of a given experimental set-up defines the principal hardware components, including distributed processors and their corresponding connections for the event data flow. This architecture can be mapped to an equivalent logical software system representation using the two software construction elements. It is then specified in a configuration file, which defines the properties of the individual stages being distributed on a set of processors in the system.

At various places in the data-acquisition system events are manipulated for various reasons such as filtering, formatting, merging, monitoring, routing, or recording. The unit



construction system allows maximization of the overall efficiency, as data buffers and processing elements are distributed in the architecture so that different levels of the chain can work concurrently on different events.

III THE CONSTRUCTION ELEMENTS

3.1 The Stage

The stage is the fundamental construction unit of CASCADE. It performs the basic functionality of a general single-processor, single-process, data-acquisition kernel. It is structured and parameterized so that several stages can be grouped together to form sub-systems such as event builders or farms.

The stage is organized in several threads of execution. This allows operations to take place concurrently on different events so that the stage can deal with a number of input and output ports at their individual rates. The thread scheduling sequence in the stage is performed on a priority basis and has no fixed sequence, as Fig. 2 indicates. This is opposed to the flow of a particular event through a stage, which passes from one thread to the next in a well defined order as shown in Fig.1. Each thread corresponds to a given operation to be performed on an event or to a control action to be done on the stage.

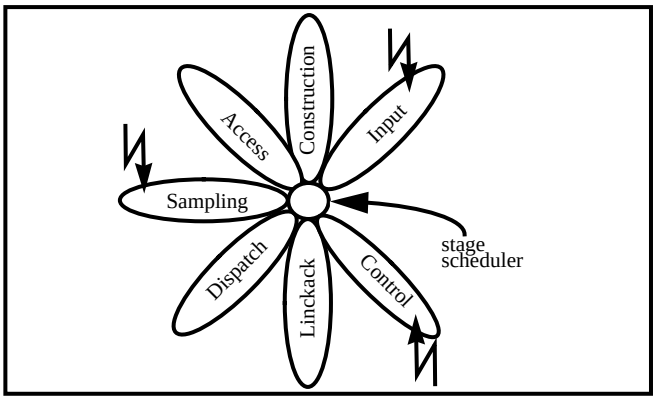


Fig 2. The stage organization into threads

3.1.1 The event data flow through the stage

Data enter the stage via one of the input connections, go through a number of possible operations, and are finally passed to other stages and to monitoring tasks which subscribed to this type of event. Threads directly involved with the main data-flow are called *phases.* The logical transition of an event through the stage phases follows a fixed sequence and is controlled via signals generated by the experiment trigger, the handshake between connected stages, monitoring requests, or internally by another thread. Events are handled and buffered in the form of event descriptors. Upon reception of a trigger event descriptors are, as shown in Fig.1, passed sequentially through:

- *the input phase*, which creates an event descriptor and, if necessary, copies the event data,
- *the construction phase*, which, in the case of an event builder, gradually links together the descriptors of the sub-events until a complete event is created,
- *the access phas*e, where they are marked for monitoring,
- *the dispatch phase* which formats and outputs them to one or several other stages,
- *the link-acknowledge phase* which marks the events to be released once the transfer through a particular output port has been acknowledged.

A set of *sampling handler phases* takes care of the connection and the event requests issued by monitoring programs which attach to the stage.

Once connected, a monitoring program issues requests for events and receives in return the relevant pointers and sizes so that it can access the event. The communication with the run-control facility is performed by the *control thread.* A scheduler has been developed to control the execution of the threads. It reacts to signals associated with each thread issued either externally by other processes or internally by one of the other threads [1].

3.2 The Inter-stage Link

Two consecutive stage units in the data-flow topology are linked by the CASCADE connection unit, the inter-stage link. Communication between two stages is initiated by the dispatch phase of the upstream stage, which triggers the input phase of the downstream stage by sending it a signal. The protocol includes exchange of a message containing the event descriptor followed by the transfer of the event data. An acknowledge message is sent by the downstream stage to the link-acknowledge phase of the upstream phase once the event has been successfully transferred.

A high-level interface and a handshake protocol have been specified for the inter-stage link communication. They have been implemented on a number of hardware and software platforms.

IV EVENT HANDLING

4.1 Event Production

Events are produced by stages which have input ports declared to be of type USER in the application configuration file. These stages are generally 'front-end' stages. They differ from the others only by the fact that some application-dependent *event production functions* have to be linked with the stage modules when the system is generated. At execution time, when the stage is triggered on one of its input ports, the input phase calls the appropriate event production functions if the input port is of type USER. These functions, for which templates are available, must read the event data and declare the event to the stage. CASCADE code and a number of I/O libraries are available to read events from a variety of busses often used in HEP experiments.

4.2 Event Building

The event builder in CASCADE is a stage which can perform in its construction phase parameterized multi-level event-building operations on the data collected from its input ports. This mode of operation has to be specified in the application configuration file together with the event types and with some dependency rules involved in the building process. A number of application-dependent *event building functions* have to be written and linked with the stage modules at system generation. These functions are automatically called by the stage to get information on the subevents and, if necessary, re-format them before performing the actual building operation.

The event-building operation can be the simple merging of subevents or events, or tagging of events with an error flag. However, event building can as well affect the whole burst. Operations such as ordering events according to their types, checking the burst consistency, verifying the presence of special data - for example, beam information delivered at every burst - can be performed. Failure of any of these checks can result in marking the burst with an error. The option to deliver one event at a time to

the output port or, instead, the whole burst as one ‘superevent’ is also available.

4.3 Local and Remote Event Monitoring

Typical CASCADE data-acquisition configurations consist of multi-crate front-end VME systems linked to a number of workstations. An important task of the workstations is to monitor and analyze the data collected by the front-end systems, which must be able to provide a high rate of events to the workstations. Monitoring on a workstation is generally preferred over event monitoring on a front-end system as it offers more CPU power and an attractive environment for monitoring tasks.

Local monitoring programs run as separate processes in the same CPU as the stage from which they retrieve events. A set of functions, provided in the form of a library, are called by the monitoring programs to connect to a given stage and specify the event-sampling criteria, to request an event, to release an event, and to disconnect from a stage.

This scheme requires the existence of a stage in each workstation which performs monitoring. In order to reduce run-control overhead and to simplify the data-acquisition configuration a new scheme called *remote monitoring* has been introduced. This allows a monitoring process (MP) to connect to a stage which is running on a remote CPU. The monitoring functions are mapped across the network in a transparent way. Monitoring programs can misbehave without affecting the stages and correct ending of the monitoring connection to the stage is guaranteed. Remote Monitoring programs may be started on any workstation with a TCP/IP connection to the target system. By loading different libraries the user can switch between local or remote monitoring mode.

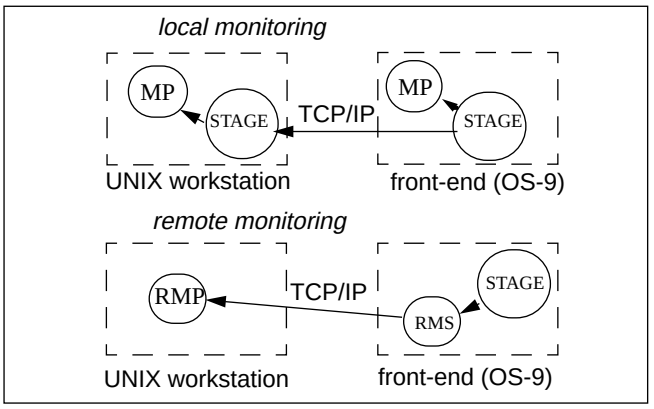


Fig 3. Local and Remote Monitoring

4.3.1 Event sampling

In the stage a service thread, called the sampling handler phase, serves the asynchronous requests issued by the various monitoring programs. It also handles the event bookkeeping in conjunction with the stage access phase.

The MP is notified asynchronously as soon as an event of the requested type becomes available. In *request mode* there is no guarantee that a minimum percentage of events will be seen. In *fixed mode* the MP is guaranteed to receive at least the percentage of events that it has specified and more if possible.

An event is made available to the MP by means of sizes and pointers to the event header, data and trailer. The communication between the MP and the stage is based on pipes for the exchange of control or (dis)connect, request and reply messages, and on shared memory for the access to events.

4.3.2 Remote monitoring

Remote monitoring is a mechanism which maps client monitoring onto equivalent server functions across the network via TCP/IP as illustrated in fig. 3. It is based on existing software elements: the local monitoring, the CASCADE network connection tool and the inetd daemon mechanism.

A remote monitoring client program (RMP) on a UNIX workstation connects to a remote monitoring server (RMS) on a processor where a stage is running. The RMP requests events from the server which - using the 'local' monitoring functions - extracts events from the stage and returns event data and status to the client, where the event is then available for analysis. The RM servers are created dynamically when a client opens a link, and disappear when the client disconnects or fails.

4.4 Event Recording

Events are recorded by a special type of stage called a *recorder* which has the unique task of storing events on an I/O device. Recorders must run in the same CPU as the stage which feeds them. Recorders have only one input and have no output to other stages. The inter-stage link between the recorder and its feeding stage is based on shared memory.

The formatting of events into blocks is performed by the dispatch phase of the stage connected to the recorder, where they are then split into fixed blocks and written to the device. Separating the formatting and actual recording over two processes permits concurrent execution of these two operations. CASCADE supports the CERN ZEBRA [4] format. At present, recording can be done on disk files, locally or via NFS, and, under OS9, it can be done either on the IBM3480-compatible STK4280 cartridge device or on EXABYTE drives.

The recorder can also be configured to use its own disk recorder server on a UNIX system accessed via the network. It allows CASCADE event data to be written to a remote disk. The same functionality as recording via NFS is provided, but the recording speed could be increased significantly. For example, when sending events from an OS9 system and recording them on a UNIX workstation the recording speed is about 500 to 600 kilobytes/s, whereas over NFS it is 30 to 40 kilobytes/s. Remote disk recording uses a TCP/IP client-server approach similar to remote monitoring.

V RUN CONTROL

The run control facility is comprised of two processes running on a UNIX workstation: the run control engine (NRC) and the human interface (XHI).

NRC is a modular, general-purpose control program allowing complex data-acquisition systems to be modeled in an object-oriented way. It is based on a system originally designed by the OPAL [5] experiment and adapted in collaboration with NOMAD to better suit their needs.

Operator interaction with the data-acquisition system is achieved by XHI, an X11/Motif program which provides a run-time configurable graphical interface including menus, dialog boxes and various types of display panels.

NRC is a process controlling data-acquisition (DAQ) units such as stages, recorders, monitoring programs and user-specific processes. Its main purpose is to provide synchronization between various DAQ units and to hold their respective states. Within NRC, each element is described in a uniform way as a Finite State Machine (FSM), as shown in Fig. 4. A FSM is an object having a predefined set of allowed states and allowed transitions between these states. Since DAQ units

are external to NRC, they are represented by internal FSM correspondents. A hierarchy of internal FSMs can be introduced to control subsets of the entire DAQ system.

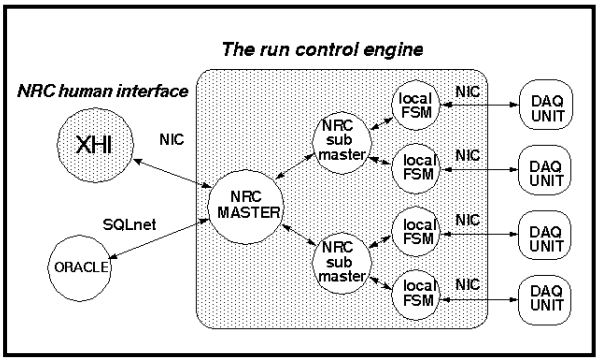


Fig 4. The NRC architecture

Communication between NRC and the external FSMs is based on NIC, the CASCADE Network component package via TCP/IP. XHI, although a special object, communicates with NRC in the same way as the DAQ units. The NRC master is the network server and the external DAQ units are the clients, which can connect dynamically to the server. NRC maintains the states of all the data-acquisition components as well as run-time parameters in an external ORACLE data base. A run control domain is defined by an identifier which is used by NRC and the connected FSM's. In a given domain each object is identified by its unique ASCII name. More than one instance of the run control facility may be running at a given time using different addressing domains. For example a full production run can coexist with the test or calibration of a particular sub-detector. When the NRC program is started the configuration information is read from an ORACLE database. This information is used to build the run control data structures: the FSM definitions, default state of objects, state transitions, elements of the graphical user interface and run time parameters. A NRC user library allows for easy preparation of an external DAQ unit using three function calls.

VI UTILITIES AND TOOLS

6.1 Error handling and reporting

A set of facilities [2] [6] is available to handle error or information messages originating from both the application specific modules and the CASCADE system code itself. It allows for message preparation outside the application code, selective message routing at run time, message transport across heterogeneous operating system platforms and support for a variety of message destination types.

6.2 The CASCADE Utility Libraries

The basic CASCADE unit and its connection unit rely on a number of other packages, each of them providing a given category of required services. Packages specifically developed for this project form 'components' of CASCADE. For some requirements, it was possible to use already existing packages.

Most of these packages are written in C and some of them in C++. Where possible they are available for all the supported platforms and hence are the pre-requisite for portable CASCADE application code.

Thread scheduling, event building, monitoring and recording services, as well as run control and error handling facilities have already been addressed earlier.

The management of event descriptors, linked lists, space allocation, and shared memory segments is provided by a specialized package. A network communication library based on TCP/IP, and one for inter-process pipe communication as well as application configuration file interpretation and debugging facilities have been written.

6.3 Development and maintenance tool

A powerful tool based on *gmake* has been developed in order to provide the CASCADE developer with a coherent set of templates, macros and symbols. It assists in the creation, validation, release, and maintenance of component packages across all the supported platforms. It also handles the distribution and archiving of the CASCADE system release.

VII SUMMARY OF APPLICATION-DEPENDENT PARTS

The application-dependent parts in a CASCADE-based data-acquisition system are:

- *event production functions* to be linked with the front-end stages;
- *event building functions* to be linked with the event builder stages;
- *monitoring programs* which may attach to stages;
- *control parameters* to be entered via SQL statements into ORACLE;
- *the configuration file*, which specifies the functional and physical characteristics of every stage and inter-stage link as well as the topology of the whole system;
- *description files* for the error handling and rooting;
- *script files* to load and start execution of the CASCADE related processes in the various processors of the application.

Templates are distributed for all the modules listed above together with template makefiles.

VIII SUPPORTED ENVIRONMENT

CASCADE relies on various other packages mainly concerning the I/O infrastructure required for physics input/output, inter-crate communication, and data recording. At present, the basic hardware and software platforms on which CASCADE is supported are as follows. The stages must execute either on the MC68040-based CES [7] FIC8234 running the OS-9 operating system in VMEbus crates or on workstations running either ULTRIX or SunOS. Triggering is done via the CES VMEbus CORBO trigger module. CAMAC LAMs or FASTBUS service requests could be considered. Front-end stages may read data directly from VMEbus or from CAMAC or FASTBUS by using the CES-supplied hardware interfaces and their associated libraries [7]. The VICbus inter-stage links and the tape recording devices interfaced via SCSI use CERN developed libraries [2], whereas communications across Ethernet for inter-stage links and

for CASCADE component communications use standard TCP/IP and NFS.

IX PRESENT APPLICATIONS

NOMAD has been the first experiment using CASCADE for its data-acquisition and has been in production since April 1994.

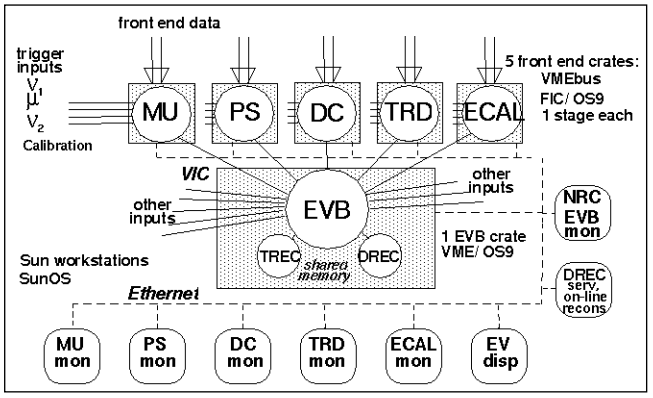


Fig 5. The NOMAD DAQ layout.

The layout consists of five 'front-end' stages, one for each sub-detector, linked via VICbus to an event builder stage. Two output ports of the EVB connect to a tape recorder stage, and to a remote disc recorder stage, all running on CES FIC8234 CPUs under OS-9. Each of the front-end stages have four input ports for three burst types and for calibration data, and one output port to the EVB. They are triggered by VMEbus CORBO modules. The event builder stage has eleven input ports: five are connected to the front-end stages while the others are special events such as beam information, calibration summary or signals for time-out and synchronization. The EVB stage makes extensive use of all the CASCADE event-building features. A number of dedicated SUN workstations running SunOS are connected to the system via Ethernet. Their tasks are to monitor the subdetector data and the event builder data remotely and to serve for run control, event display, on-line reconstruction, disk recorder server and as OS9 boot and file server. During the first months of operation of CASCADE some instabilities were experienced. After consolidation and tuning, the system is now operating smoothly and fulfilling all its requirements.

The Energy Amplifier Project [8] has also used CASCADE during 1994. Its data-acquisition system was simpler and included one stage and one recorder. However, it had to handle very large events of up to four Megabytes at a low rate. The data-taking went very successfully.

X PERFORMANCE

At present, the stage has an overhead of one millisecond on a FIC8234 running OS-9. Parameters such as event size, system architecture, type of recording, number of monitoring programs, as well as their sampling criteria, have a strong influence on the overall performance. Currently, the CASCADE-based applications are limited to several hundred events per second in continuous mode and several thousand events per second in burst mode. A systematic time analysis coupled with the optimization of the critical areas in the code is to be initiated.

XI FUTURE DEVELOPMENTS

Several experiments and projects have recently announced either their intention or their decision to use CASCADE for the data taking of their experiment or for their tests. RD24 [9] is to demonstrate the use of the Scalable Coherent Interface (SCI) in a data-acquisition environment together with parallel event building using CASCADE. The CES Fast Data Link (FDL) may be evaluated in a similar environment. Both of these requests have initiated new developments aiming at the support of SCI and FDL as inter-stage link media and at the possibility of handling farms of stages in particular for event-building. Porting of CASCADE to new platforms has started. The port to the LynxOS operating system is well advanced whilst OSF/1 is being considered. The use of PowerPC-based processors will start after some evaluation of the market offerings. A Digital Linear Tape device (DLT) will be proposed as a recording device for CASCADE-based applications.

XII CONCLUSIONS

The design concept of a unit construction system for a data-acquisition system used in HEP experiments has shown strong advantages at various levels. The logic system layout feature, easy task distribution in a heterogeneous environment while keeping a homogeneous concept, and scalability aid the experimenter. Recent additions have proven that the integration of new platforms and hardware connections is straightforward. The emphasis on modularity and the design concept of a unit construction system allow new user requirements or advances in technology to be accommodated without design changes.

XIII ACKNOWLEDGEMENTS

We would like to thank the NOMAD on-line group for their effort in checking out CASCADE and their useful suggestions for enhancements. The usage of CASCADE by the Energy Amplifier project has stimulated a number of improvements as well. OPAL has kindly provided the VICbus message library and the core of the error handling system and of the run control. NOMAD has contributed to the adaptation of the run control to CASCADE. We are grateful to the ESS/OS group for their contribution in testing and integrating the hardware and to the CN-CE group for their assistance with OS-9 and LynxOS issues.

XIV REFERENCES

[1] CASCADE: A Toolkit for the construction of distributed, real-time, data-acquisition systems, Perrin,Y. et.al., Conf. Record of RT93,Vancouver, Canada, 1993.

[2] CERN ECP/DS, CASCADE User Guide, internal notes, http://cascade.cern.ch/Projects/Cascade, Geneva.

[3] NOMAD: WA96 Proposal, CERN-SPSC/P261,1991, Geneva, Switzerland.

[4] ZEBRA: CERN Program Library Long Write-up Q100/Q101, Geneva, Switzerland.

[5] The Object Oriented DAQ Control Program of the OPAL Detector at LEP, Stephen Wotton, Proceedings of CHEP-92, Annecy, France, 1992.

[6] EMUX - Error Message Utility for OS-9 and POSIX, Meijers, F., CERN ECP/DS, January 1992.

[7] Creative Electronic Systems SA, Geneva Switzerland.

[8] Experimental determination of the energy generated in nuclear cascades by a high energy beam, Physics letters B 348 (1995) 697-709, 6.April 1995

[9] RD24: Status report CERN/DRDC/94-23, May 1994, Geneva, Switzerland.