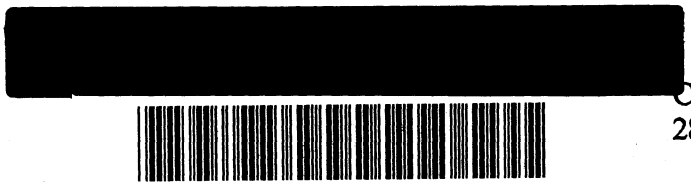
EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH



CERN/ECP 93–13
28 October 1993

CM-P00049301

CASCADE: A TOOLKIT FOR THE CONSTRUCTION OF DISTRIBUTED, REAL-TIME, DATA-ACQUISITION SYSTEMS

Y. Perrin, C. Bizeau, W. Bozzoli, D. Burckhart, Y. David, M. Gentile, J-P. Matheys, J. Petersen, L. Tremblet, P. Vande Vyvre, A. Vascotto and P. Werner

CERN, Geneva, Switzerland

Abstract

CASCADE is a software package developed at CERN for the construction of distributed, real-time, data-acquisition systems for high-energy physics (HEP) experiments. It aims at providing a high degree of homogeneity between all levels of the acquisition chain and it allows the building of a wide range of system configurations. CASCADE is mainly written in C, with some class libraries in C++. The first implementation supports configurations based on VMEbus processors running OS-9 and UNIX workstations interconnected via VICbus and Ethernet.

Presented at the Eighth Conference on Real-Time Computer Applications in Nuclear, Particle and Plasma Physics, Vancouver, Canada, 8–11 June 1993

1 INTRODUCTION

The structural and functional analysis of various high-energy physics (HEP) experiment data-acquisition systems shows that these systems have a great deal of commonality. Whereas major differences reflect the specificity of the physics studied and of the detector techniques used, other differences are due to practical constraints, such as budget, laboratory standards, and technology limitations at a given time.

Starting from these observations, the data-acquisition group of the ECP Division at CERN is currently developing a new data-acquisition system called CASCADE (CERN Architecture and System Components for an Adaptable Data-acquisition Environment). CASCADE is designed to provide physicists with a set of software building blocks so that they can construct their data-acquisition system in a more homogeneous way.

2 ARCHITECTURE

Most data-acquisition systems in HEP can be viewed as multiprocessor, tree-structured architectures. Data flow in the system under the form of *events* which group information related to a given trigger. All events progress in the tree in the same direction. In their migration, events are accessed for various reasons, such as filtering, formatting, merging, monitoring, routing, etc. These operations are performed by processes running in a set of processors distributed in the architecture. Depending on their role and on their physical environment in the configuration, these processors may be of different types, may run different operating systems, and may be linked by a variety of inter-processor links as shown in Fig. 1. To maximize the overall efficiency, buffering and intelligent entities are distributed in the system so that different levels of the chain can work concurrently on different events.

To a large extent, CASCADE allows the physicist to concentrate more on the functional aspects of the data acquisition than on its physical implementation. This is achieved by grouping the most common functions in a single building element called a *stage*, which can be replicated at all levels and which has been implemented on a number of hardware and software platforms widely used at CERN.

Building a distributed data-acquisition system with such an approach consists of the following steps:

- making a functional representation of the application requirements based on stages as shown in Fig. 2;
- determining the hardware and software platforms to be used at the various levels;
- writing a *configuration file* to specify the topology of the system and the mapping between the logical and the physical representation as illustrated by Fig. 3;
- writing the application-specific code to be linked with the CASCADE software building blocks and with the appropriate I/O libraries.

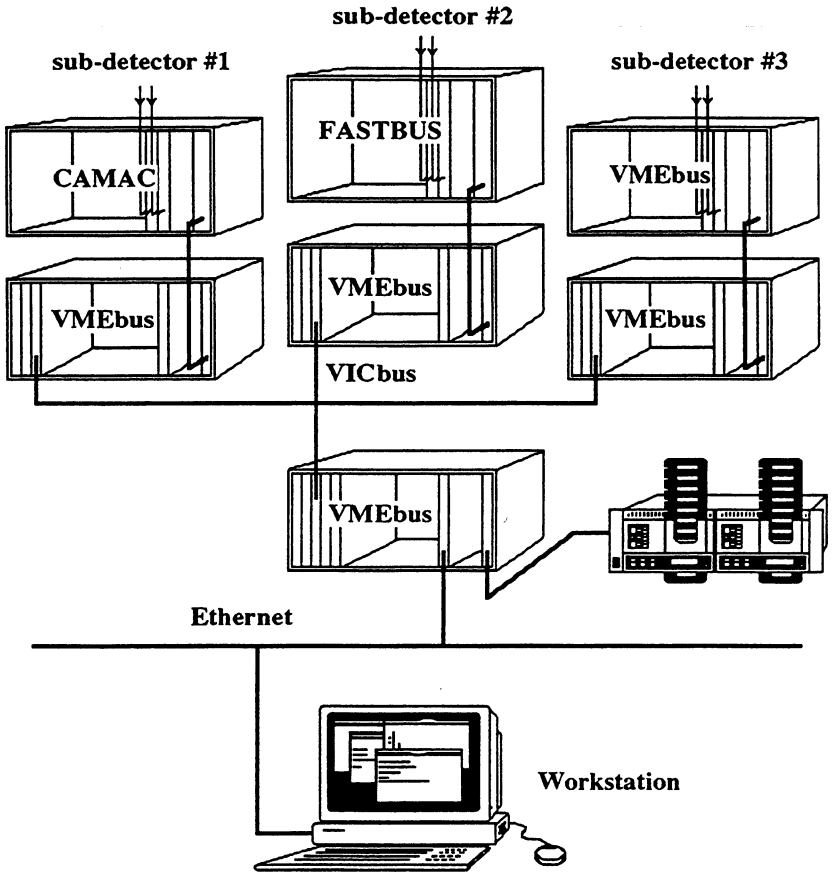


Fig. 1 The data-flow architecture of a HEP experiment readout system

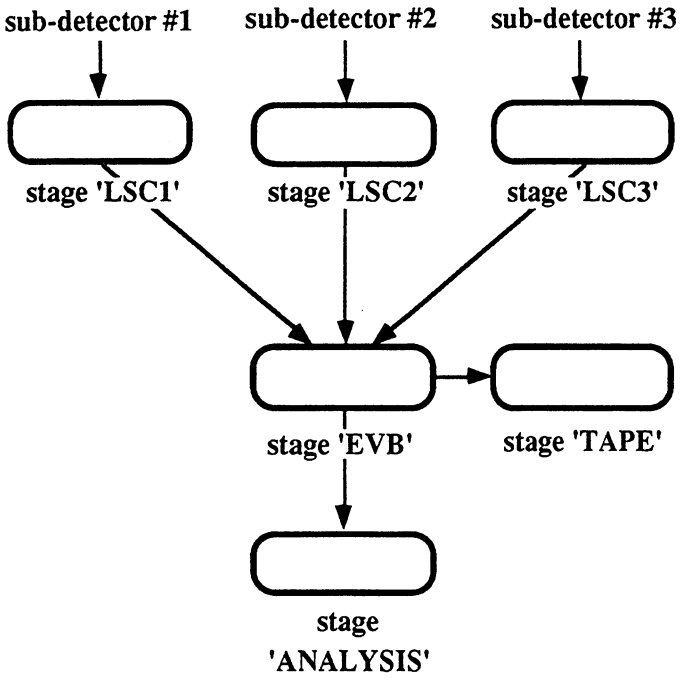


Fig. 2 CASCADE representation of a data-flow architecture

```
STAGE_NAME = LSC1
        CONSTRUCTION_TYPE = DUMMY
        NB_INPUTS = 1
        NB_OUTPUTS = 1
      INPUT1
        CONNECTED_STAGE = NONE
        LINK_TYPE = USER
      END_INPUT
      OUTPUT1
        CONNECTED_STAGE = EVB
        LINK_TYPE = VIC
        FORMAT = RAW
      END_OUTPUT
END_STAGE
                .
                .
STAGE_NAME = EVB
        CONSTRUCTION_TYPE = EV_BUILDER
        NB_INPUTS = 3
        NB_OUTPUTS = 2
      INPUT1
        CONNECTED_STAGE = LSC1
        LINK_TYPE = VIC
      END_INPUT
                .
                .
      OUTPUT1
        CONNECTED_STAGE = TAPE
        LINK_TYPE = SH_MEM
        FORMAT = ZEBRA
      END_OUTPUT
                .
                .
END_STAGE

STAGE_NAME = TAPE
        CONSTRUCTION_TYPE = DUMMY
        NB_INPUTS = 1
        NB_OUTPUTS = 0
      INPUT1
        CONNECTED_STAGE = EVB
        LINK_TYPE = SH_MEM
      END_INPUT
END_STAGE
                .
                .
```

Fig. 3 Elements of a configuration file

3 THE CASCADE BASIC BUILDING ELEMENT

The stage is the basic building element of a CASCADE-based data-acquisition system. It can be replicated at all levels and stages can be connected together via *interstage links.*

3.1 The stage

A stage has the basic functionality of a general single-processor, single-process, data-acquisition kernel. It is structured and parametrized so that several stages can be grouped together to form subsystems, such as event builders, farms, etc. As far as the data flow is concerned, a stage could be viewed as a pipeline with one or more inputs and one or more outputs as shown in Fig. 4.

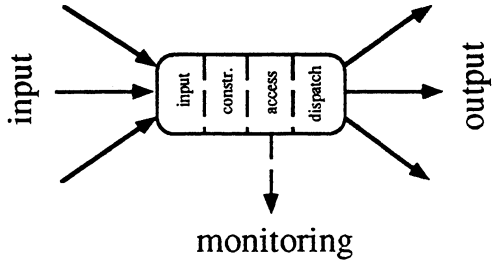


Fig. 4 A stage

Upon reception of a trigger the stage inputs an event, performs a number of possible operations on this event, and finally passes the results to other stages and to monitoring tasks that subscribe to this type of event. Inside the stage, events are handled and buffered in the form of event descriptors. The event data are copied in the stage only if strictly necessary. To minimize the dead time and to allow operations to take place concurrently on different events so that the stage can deal with different input and output rates, the stage is organized in several threads of execution. Each thread corresponds to a given operation to be performed on an event or to a control action to be done on the stage. Threads directly involved with the main data flow are called *phases*.

In a stage, events transit 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 subevents until a complete event is created.

From there, events are pushed to:

- the access phase where they are made available for monitoring;
- the dispatch phase which formats, and outputs them to one or several other stages.

A scheduler has been developed to control the execution of the threads. It reacts to signals associated with each thread and issued either externally by other processes (stages, monitoring programs, control programs) or internally by one of the other threads. If necessary, a thread can suspend its execution until a global shared resource (e.g. memory space) has been released. Thread scheduling is done on a priority basis.

3.2 The inter-stage link

Two consecutive stages in the data-flow topology can be linked physically in a number of ways. A high-level interface and a handshake protocol have been specified, and implementations have been carried out for a number of hardware and software platforms widely used at CERN. This approach provides homogeneity in the overall system. It makes the communication between stages transparent to the application-dependent code. At present, implementations exist allowing two stages running in the same processor to be linked via shared memory. If the stages run on different processors they can be linked via VICbus [1]. Communication over Ethernet is being implemented using TCP/IP.

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, possibly followed by the transfer of the event data if the type of link makes it necessary or if the user has explicitly specified it in the configuration file. An acknowledge message is sent by the downstream stage once it is ready to work on the event.

4 EVENT PRODUCERS

Events are produced by the 'front-end' stages. These stages 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 event production functions if the input port is declared to be of type USER in the application configuration file. These functions, for which templates are available, must read the event data and declare the event to the stage. CASCADE and a number of I/O libraries, listed later in this paper, are available to read events from a variety of buses often used in HEP experiments.

5 EVENT CONSUMERS

5.1 Recording

Recording is done by a special type of stage called a *recorder*, which has as a unique task the storage of events on a physical device. Recorders have only one input and have no output to other stages. The interface and the protocol used to communicate with a recorder are the same as for inter-stage communications. As opposed to a true stage, a recorder does not handle true events. The dialogue between a stage and a recorder concerns events which, in fact, are fixed-length portions of one physics event or groups of several events which correspond to physical records on the storage device.

The organization of events into fixed blocks is actually done by the dispatch phase of the stage connected to the recorder. Splitting the formatting and the actual recording over two processes permits concurrent execution of these two operations. CASCADE supports both the CERN ZEBRA and EPIO formats. At present, recorders exist for the EXABYTE and for the IBM3480-compatible STK4280 cartridge device under OS-9. The development of a recorder to store the data on disk files (locally or via NFS) is planned.

5.2 Monitoring

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, is called by the monitoring programs to connect to a given stage and to specify the event-sampling criteria, to request an event, to release an event, and to disconnect from a stage. There are two modes of sampling. In the *request* mode the monitoring program receives an event as soon as there is one available but with no guarantee that a minimum percentage of events will be seen. In the *fixed* mode the monitoring program is guaranteed to receive at least the percentage of events that it has specified and more if possible. In both modes the monitoring program is notified asynchronously as soon as an event of the requested type becomes available.

An event is made available to the monitoring program by means of a pointer. The monitoring program does not know whether the event data is local to the stage or remote in a previous stage. The space occupied by the event is automatically locked and must be explicitly released as soon as the monitoring program has finished with it.

The communication between the monitoring programs and the stage is based on pipes for the exchange of request and reply messages, and on shared memory for access to events. In the stage a service thread, called the sampling handler, serves the asynchronous requests issued by the various monitoring programs. It also handles the event bookkeeping in conjunction with the stage access phase.

6 RUN CONTROL

The CASCADE control involves a process running in a workstation, server processes in the target processors, and a control thread in the stages. The run-control program is application-independent. The human interface is based on MOTIF. A set of menus is used to specify a number of run parameters and to control the data taking. On selection of a menu item, commands are sent to the appropriate stages according to an application-dependent *control file.*

A status display option is available via a menu. When selected it presents the status information of all the stages involved in the application in a synthetic way.

The communication between the run-control program and the various stages is based on messages consisting of strings of characters. Routing and asynchronous delivery of these messages to the destination processors use a commercially available message system called ISIS [2].

7 INFRASTRUCTURE

As mentioned earlier, CASCADE provides the user with building blocks and tools so that the data-acquisition architecture can be viewed and handled in a homogeneous way. This goal has been achieved by the specification and the development of high-level elements which are, to some extent, platform-independent. In addition, an infrastructure of I/O and system libraries is necessary to adapt to the various hardware and software environments.

7.1 The BLC library

The suitability of object-oriented techniques in a real-time environment has been assessed by implementing some basic services as a C++ class library called BLC (Buffer and List-handling Classes) [3]. These services include the management of event descriptors, linked lists,

space allocation, and shared memory segments. An object display program has been developed to help with debugging. Implementations of the BLC class library and of the program exist for ULTRIX and for OS-9.

7.2 The I/O libraries

The first applications of CASCADE will be performed on a multicrate, multiprocessor, VMEbus-based system connected to workstations. The CPUs in the VMEbus crates — currently the MC68040-based CES FIC8234 [4] — will run the OS-9 operating system, whilst the workstations will run either ULTRIX or SunOS.

In order to implement a multistage CASCADE configuration on such systems, a hardware and software infrastructure is required for physics input/output, intercrate communication, and data recording.

Front-end stages may read data from CAMAC, Fastbus, or VME. Interfacing to CAMAC is provided via a VME-to-CAMAC branch interface [5] or via a dedicated CAMAC controller connected to VICbus [6] with software support in the form of standard NIM/ESONE/IEEE libraries. A Fastbus-to-VSB interface [7] provides the connection between a VME crate and Fastbus. A comprehensive multiuser implementation of the NIM Fastbus library has recently become available for this module [8]. Triggering is possible from CAMAC (lams), Fastbus (service requests), and VMEbus (via a general-purpose VME trigger module called CORBO [9]).

To provide a fast channel for data transfers and for the exchange of messages, the VME crates are interconnected via memory-mapped VICbus links [1]. Ethernet is used for less-time-critical applications, such as run control and file access, via NFS.

The inter-stage communication across VICbus is based on a message exchange library developed at CERN [10], whilst the communication across Ethernet uses the standard NFS/TCP/IP package available on both OS-9 and UNIX. Under OS-9, data may be recorded, via SCSI, either on an IBM3480-compatible STK4280 tape drive [11] or on an Exabyte [12].

7.3 The EMU library

Error reporting is based on the CERN-developed Error Message Utility [13, 14] which is currently adapted to using TCP/IP for network communications.

8 APPLICATION-DEPENDENT PARTS

In summary, the application-dependent parts in a CASCADE-based data-acquisition system are:

- the event production functions to be linked with the front-end stages;
- the monitoring programs which may attach to stages;
- the configuration file which specifies the functional and physical characteristics of every stage as well as the topology of the system;
- the control file which describes the sequence of commands to be sent to stages for each control action.

9 STATUS

A prototype system has been built and demonstrated. This system was made of two stages linked via VICbus and a tape recorder, all of them running on CES FIC8234 CPUs under OS-9. The front-end stage was triggered by a VMEbus CORBO module and the recording device was a STK4280 cartridge tape. The system was controlled from a DEC workstation running ULTRIX.

This first test was essentially to check the validity of the overall approach and to verify the logic of the CASCADE elements which are still in a preliminary form. The project is continuing, in particular, in the areas of event building, error handling, run control, product distribution, and system generation. Performance analysis and optimization are still to be done.

10 FIRST APPLICATION

A first prototype of the system will soon be given to the CERN WA96 experiment. This first implementation will use VMEbus-based processors running OS-9, VICbus for the inter-crate data transfer, and a cluster of Sun workstations loosely coupled to the OS-9 systems via Ethernet.

The complete system to be made generally available is expected to be ready by the end of 1993. Support of CASCADE under LynxOS (as requested by the CERN NA48 experiment) is foreseen in the longer term.

References

[1] Creative Electronic Systems SA, VIC 8251F user's manual, version 0.1, Geneva, Switzerland.

[2] ISIS Distributed Systems, Inc., ISIS user reference manual, version 3.0, Ithaca, NY, USA (1992).

[3] P. Vande Vyvre, Buffer and list handling classes, CERN/ECP–DS group note (1992).

[4] Creative Electronic Systems SA, FIC8234 user's manual, version 0.5, Geneva, Switzerland.

[5] Creative Electronic Systems SA, CBD8210 CAMAC branch driver user's manual, Geneva, Switzerland.

[6] Creative Electronic Systems SA, VCC 2117/A CAMAC crate controller user's manual, version 1.1, Geneva, Switzerland.

[7] Creative Electronic Systems SA, FVSBI 9210 Fastbus-to-VSB interface user's manual, Geneva, Switzerland.

[8] Creative Electronic Systems SA, FVSBI Fastbus library, Geneva, Switzerland.

[9] Creative Electronic Systems SA, RCB8047 CORBO VME readout control board user's manual, Geneva, Switzerland.

[10] F. Meijers, and J. Petersen, A VME intercrate message system based on VICbus and OS-9, CERN/ECP–DS group note (1992).

[11] J. Petersen, A 'high-level' driver for the STORAGETEK STK4280 (SUMMIT), CERN/ECP–DS OS-9 SCSI2 group note (August 1991).

[12] J. Petersen, A 'high-level' driver for the Exabyte, CERN/ECP–DS OS-9 SCSI5 group note (August 1991).

[13] P.C. Burkimsher, EMU — the MODEL error message utility, CERN/ECP–DS group note (December 1990).

[14] F. Meijers, EMUX — error message utility for OS-9 and POSIX, CERN/ECP–DS group note (January 1992).