ALEPH 93-171
DATACQ 93-002
M. Cattaneo et al.
2.11.1993

The new slow control system for the ALEPH experiment at LEP[1]

P. Mato, A. Engelhardt, J. Harvey, G. Mourouvapin, M. Saich, W. Tejessy

CERN, European Organization for Nuclear Research, Geneva, Switzerland

W. Cameron, M. Cattaneo

Department of Physics, Imperial College, London, UK

R. Fantechi, E. Mannelli

Dipartimento di Fisica dell' Universitá, INFN Sezione di Pisa, e Scuola Normale Superiore, Pisa, Italy

E. Veitch

Department of Physics, University of Edinburgh, Edinburgh, UK

O. Callot

Laboratoire de l'Accélérateur Linéaire, Université de Pais-Sud, Orsay, France

The ALEPH slow control consists of 7000 channels in G64 distributed over 35 networked microprocessors which are used to control and monitor the experimental apparatus, such as high voltage and gas systems To improve performance, the readout has been upgraded from a ROM based system to use 3U VME processors, running OS9, with a G64/VME interface. A new object oriented design for the software has been implemented, where the full description of the system is held centrally in a relational database on the host VAX cluster. Control and monitoring is carried out through a library which accesses the database and handles communications with the processors over ethernet using a client-server model. The design and implementation of the system and initial experience with its use are described.

1 Introduction

The Aleph detector at LEP is composed of a dozen independent sub-detectors, located in a cavern 140 meters below ground, which are operated remotely from a control room on the surface. This is achieved by a slow control system which controls and monitors about 30 different types of hardware devices (temperature and pressure sensors, electronics crates, high voltage systems, etc.) via some 7000 channels distributed around the detector.

The original slow control system of Aleph is described in [1]. The hardware available in the early 1980's, when this system was built, placed severe constraints on the performance of the system. The individual channels were monitored by software which, due to memory limitations, had to be burned into the EPROM of the G64 microprocessors connected to several UTI-NET segments around the detector. The UTI-NETs were connected to the Ethernet of the Vax cluster in the control room via gateways. A server and a database on the Vax translated the high level commands issued by the application software of the sub-detectors into the small set of low level commands understood by the micros, and routed them to the appropriate micro. Low level replies in the opposite direction were handled in a similar way by the same server. While this system operated satisfactorily under stable conditions, the bottlenecks introduced by the gateways and the server task caused

[1] Paper presented at the International Conference on Accelerator and Large Experimental Physics Control Systems, Berlin, October 18-22, 1993

unacceptable delays when several simultaneous control actions were required, or when several alarms were received.

In 1992 it was decided to replace the old G64 processors with 3U VME processors interfaced to G64 running OS9 and connected directly to the Ethernet of the experiment. This presented a unique opportunity to re-engineer some of the software layers, in particular the system software used as building blocks for the control applications. In this paper we describe the requirements, design and implementation of this new software.

2 Functional Requirements

Given the diversity of equipment and the large number of people from different groups responsible for its operation, a basic goal was to design a homogeneous system with a uniform architecture. At the same time, this architecture has to be flexible to satisfy the real-time requirements of individual subdetectors. These requirements lead naturally to a client server model where the role of application tasks is to formulate requests to perform actions on the slow control system and for the system software to execute those requests. The server functions, control and monitoring, are cpu intensive and can be implemented using a distributed system of microprocessors. The software should be designed and implemented in such a way that, using this client server model, the load on the host computers and networks is minimized.

It is important that the interface between the application (i.e. client) and the system (i.e. server) should be as simple as possible, which can be achieved if the basic object on which an operation can be performed has a high level of abstraction. The concept of a slow control device has therefore been introduced, all slow control actions being operations on devices. An example of a slow control device is a Fastbus crate. It can be turned on or off and its state can be monitored such that an alarm is generated if the current state does not agree with the desired value. Another example is a temperature sensor which can be monitored to compare the analogue reading with a nominal value. Again an alarm should be generated if the current value drifts outside acceptable limits. In both these examples the internal features of the device, i.e. the number and characteristics of the digital and analogue channels used to implement the functions available, are deliberately hidden from the application.

Another requirement is that it should be easy to adapt the system software to reflect changes in the slow control hardware. For example, it is important that the data representation used to describe system components should be easy to modify as devices are added to, or removed from, the system. The software structure should also be able to accommodate new functions, corresponding to operations on new types of devices.

Practical considerations lead to further requirements. It should be possible to recover from failures, such as power cuts and program crashes. In particular such events should not result in a flood of alarm messages which can easily cause the system to "hang".

3 System architecture

Due to the decomposition of the Aleph experiment into sub-detectors it seems natural to decompose also the slow control system into independent control and monitoring applications in charge of single sub-detectors. Each of these sub-detector control applications uses the slow control system software which is common to all of them. This allows us to satisfy the individual requirements for each sub-detector while assuring the coherency and homogeneity of the system.

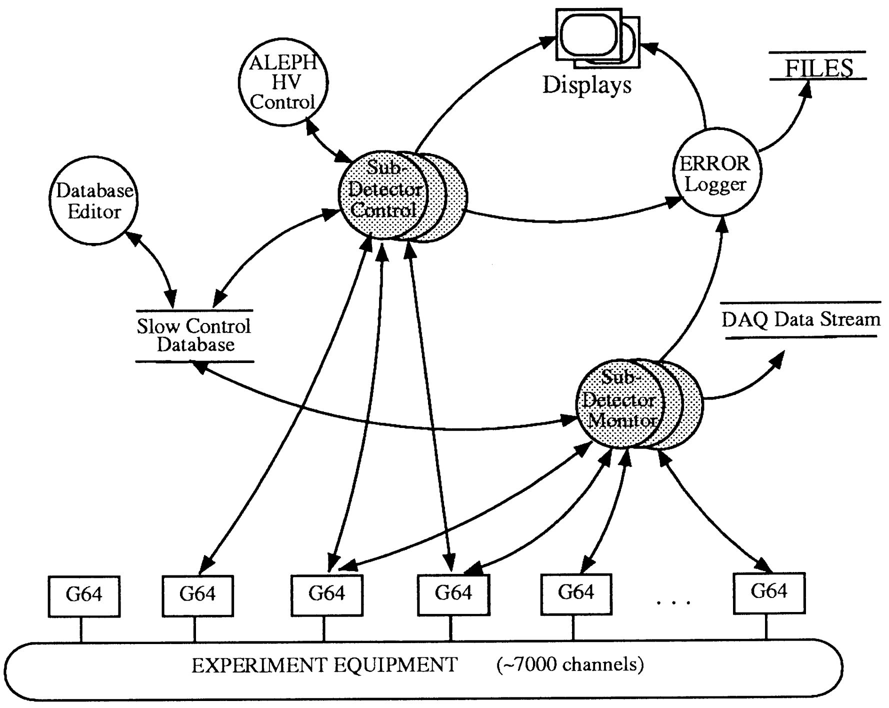

Fig1: Overall view of the ALEPH slow control system

Figure 1 shows an overview of the task architecture for the slow control system. We can see how each sub-detector control/monitoring application makes use of the common services such as a global database, an error reporting and logging system, a slow control system software to perform control and monitoring actions on the physical devices. Also we must provide control applications which are one level above in the control hierarchy to allow the operation of the experiment as a whole. A typical example of this is the centralized high voltage control.

Object Oriented programming (O-O) techniques have been used to design the slow control system software. This is because it is very natural to think of the slow control devices introduced in section 2 as *Objects*. In this way, we can easily hide the internal details of the device from the user application. To perform an action on a device, the control application needs first to create a device object using the identifying name and then to call the member function which forwards the request to a server which performs the desired action. When the device is created, it gets its parameters i.e. addresses, number of channels, channel descriptions, etc. from a relational database.

The slow control database contains the complete description of all the devices in the experiment. It is not a truly O-O database, but it has the O-O feature of allowing the definition of new types of devices as they are needed. The system software does not need to be changed when a new type of device is added into the system. The description of a device type is done by defining the static data structures which are needed to fully describe the device, the dynamic data structures which are used when interacting with it, and the actions which are allowed for each device of this type.

The client-server model has been fully exploited in the design (Figure 2). The user control application is at one end of the client-server chain and the channel I/O library is at the other end. The only contact point between the user application and the system software is the "device" class and also, if necessary, the "list of devices" class. This list class has been introduced to improve performance when identical actions have to be performed on many devices at once. The command server object is in charge of executing device action requests in a synchronous way. A possible device request is the command to start monitoring a given device. This is translated to an insertion of the device information into a shared local database, which is then used by the monitor and alarm handler to perform the monitoring function.

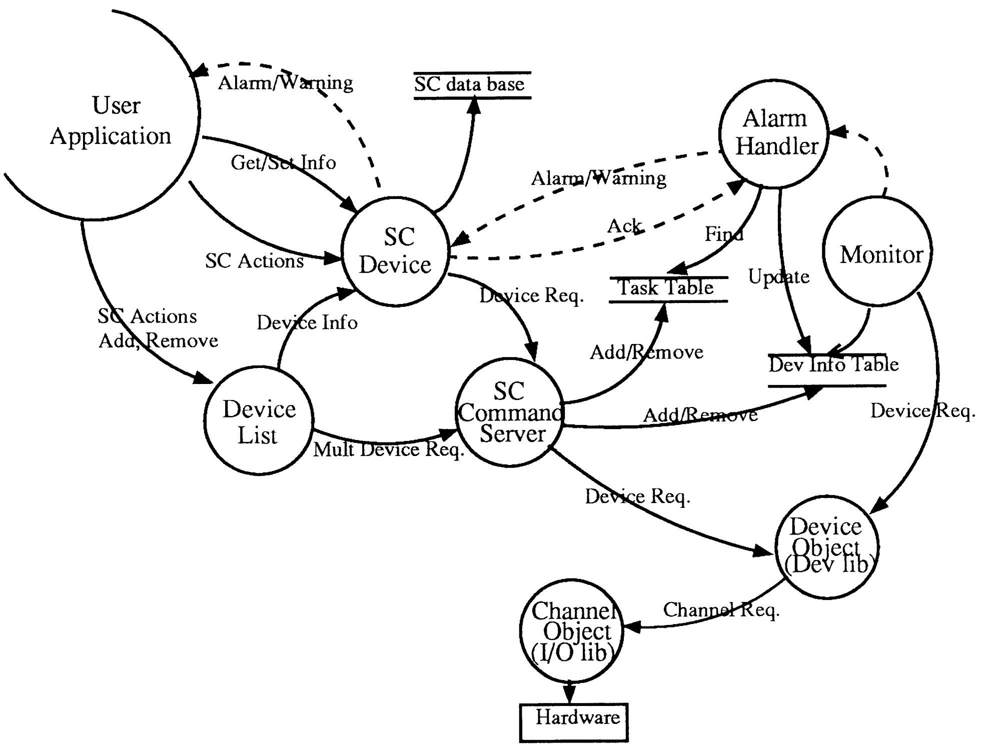

Fig 2: Client-server diagram showing the main objects (circles) together with the synchronous (solid arrows) and asynchronous (dashed) messages exchanged between them

An effort has been made to encapsulate the device type specific commands into a single place in the system, i.e. a set of device drivers called the device library. These drivers are simple subroutines calling a number of channel I/O routines. The device library is the only piece of code which needs to be upgraded if new device types are introduced into the system.

4 Implementation of the system software

The command server, monitor, and alarm handler objects have been implemented as separate programs running in the microprocessors (VM20 from PEP modular Computers) under the OS9 operating system and written in C. The three programs are clients of the device library, which has been implemented as a multi-user library, thus allowing concurrent access to the hardware. The OS9 operating system supports the TCP/IP transport protocol which is used to implement the communication between the clients and the servers. Features of OS9 like time sharing with priority, resource locking, shared memory are used extensively.

The VAX/VMS operating system is used on the on-line computers. The few VMS dependencies in the system software have been encapsulated into objects. For example, the implementation of the database uses a VMS cluster-wide global section, but a different implementation can be easily produced without changing the design at all and only affecting a single module. The system software of the VAX is written using the C++ language. However a FORTRAN and C interface has been provided to allow sub-detector applications to be written in any of the three languages.

A specialized database editor has been developed to enter all the information needed to describe the slow control system: crates, devices, device types, etc., as well as the operational parameters such as warning and alarm limits, desired states, etc. The editor can generate the include file containing the definition of the data structures and parameters for each device type. This include file is then used at one end by the user applications and at the other end by the device drivers on the microprocessors.

The manpower required for the design and the implementation of the new slow control system software has been about 18 man-months.

5 Conclusions

During the design of the software system, formalized review sessions were regularly held among the system designers and applications programmers to ensure that the requirements which were very well known due to experience with the previous system were met. This has led to a substantial reduction in the time required to adapt existing applications and new ones into the overall control system. By limiting the number of controllable objects to a single hierarchical layer of named devices and by providing a user-friendly database editor to incorporate all parameters pertaining to each device, the writing of application tasks has become much simpler than in the former version, making the code easy to read and therefore enhancing maintenance.

The changeover in several steps, while leaving elements of the old system in place, and several months of operational experience have shown the new system to be fast, flexible and reliable. A performance of ~100 transactions/sec. can be obtained when using a single server. This performance is substantially increased when using the concurrent access possibilities to many servers at the same time. For example, the read-out of some 500 parameters from 12 G64 crates of a sub-detector can be done in less than a second. During the operational period, new device types have been added without perturbing the running system, and general reliability has been strongly improved.

6 References

1. T. Charity, et al. "The ALEPH slow control system", Proc. Computing in High Energy Physics 1990 Santa Fe