ALEPH 90-90
DATACQ 90-9
A. Belk, D.R. Botterill, J. Harvey
June 18, 1990

Contributing Paper for the **International Workshop on Software Engineering, Artificial Intelligence and Expert Systems for High Energy and Nuclear Physics**, Lyon March 1990

The Design and Implementation of a Database for ALEPH DAQ System

The ALEPH experiment required a highly flexible Data Acquisition (DAQ) System. A large number of parameters are needed to describe the system states in a complete and consistent way. These must be available to many parts of the software. The design, development and implementation of the DAQ databases are discussed, with emphasis on common access packages and the exploitation of features of the host computer.

The Design and Implementation of a Database for the ALEPH DAQ System

A.Belk

Imperial College
London, UK

D.R.Botterill, J.Harvey

Rutherford Appleton Laboratory
UK

The ALEPH experiment required a highly flexible Data Acquisition (DAQ) System. A large number of parameters are needed to describe the system states in a complete and consistent way. These must be available to many parts of the software. The design, development and implementation of the DAQ databases are discussed, with emphasis on common access packages and the exploitation of features of the host computer.

1. Introduction

ALEPH [1] is a large experiment, with a complex DAQ system. It must be flexible, fault-tolerant, robust and conceptually simple to use. Activities such as hardware and software debugging, data-taking, calibration and software development are necessary, all with differing environmental requirements [2]. Future detector configurations must be foreseen (with the minimum of software modifications). There are a vast number of system states and descriptive variables to handle, including the readout and control sub-systems, and the configuring and management of data-taking activities.

To satisfy these criteria, a system based around a sophisticated database has been implemented.

2. Design Approach and Requirements

The detector has a hierarchial structure. The levels are detector hardware, digitisers, readout modules, event builders, sub-detectors, partitions and ALEPH. The operator can specify a partition by name, (any self-contained sub-set of the system capable of data-taking) and all the levels below are automatically configured using the information held in the system database. This detail is managed by specialists and need not be known by the operator.

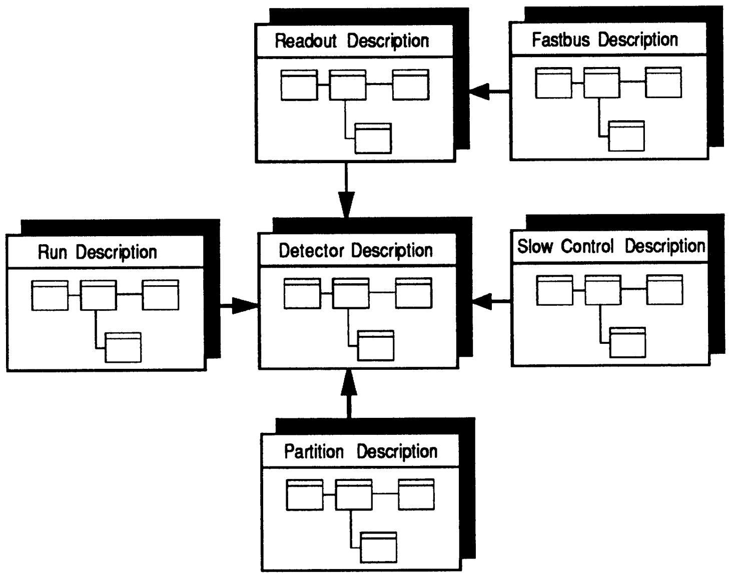

Figure 1: System Database Context

The system naturally segments into a number of components. Their connectivity can be represented in a relational database structure, and the Entity Relationship technique was adopted for the data modelling [3]. The ER Diagrams were translated to ADAMO [4] Data Definition Language. The generation of a Data Dictionary from this source provided a consistency check both within the online software, and with offline. The integrity of the model is also assured, and redundancy can be controlled.

Each component describes one level in the hierarchy or a different system function (see fig.1). Central to the scheme is the abstraction of the detector. It is described in a database and can be modified, thus in principle the scheme is not experiment-specific. Fig.2 shows the hierarchial structure of the detector and how the readout sub-system is organised on similar levels.

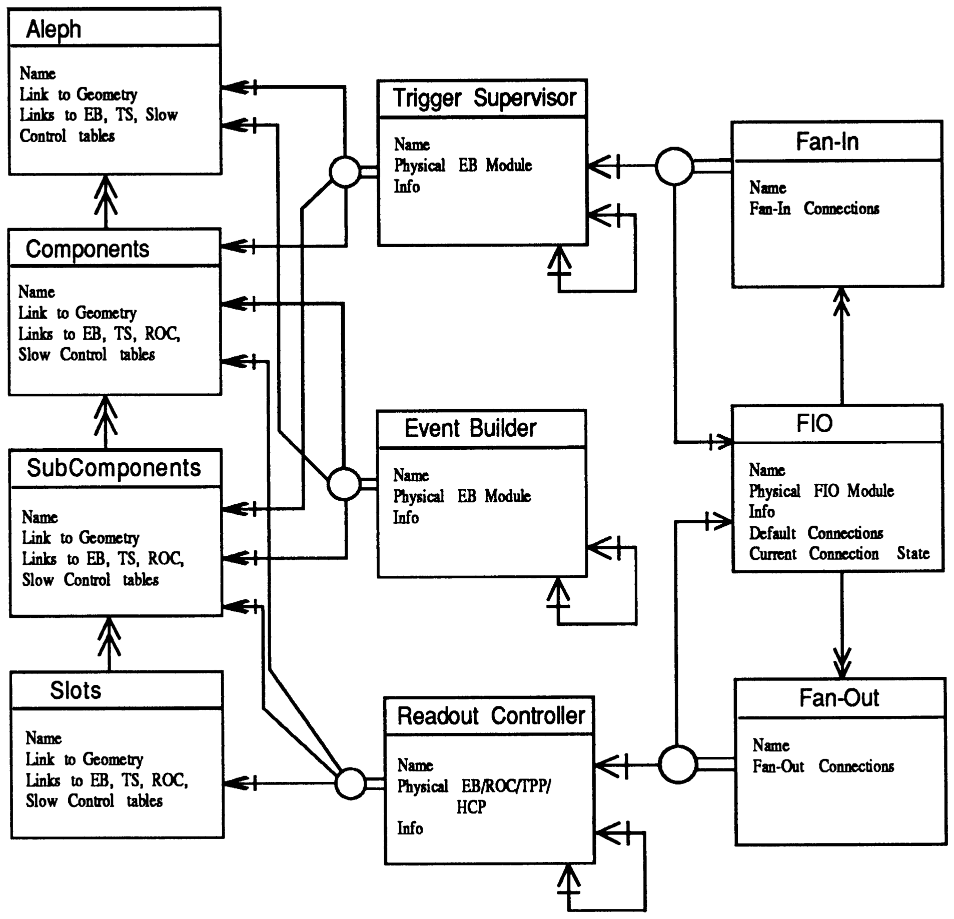

Figure 2: Detector and Readout Description ER Diagram

From the user's point of view, a partition is a collection of parts of the detector (fig.3). This determines the source of the data which will be used offline. From the partition, which parts of the readout to use is determined. This leads, in turn, to a list of physical devices to be read out, and a separate description for each node in the readout tree, specifying the sources and destination of data. The paths for trigger signals are defined by the FIO connections to the terminal nodes, or ROCs, and are programmed. The details of this, the system task architecture and the software and hardware protocols have been presented earlier [5]. All actions are derived from the initial partition name.

The readout section details the logical function of each object in the readout chain; here it is important to note that this may be performed by any physical module. For example, if a Fastbus module is changed, the procedure will run as before, but the readout database will point to a different physical device. The result is the new device performing the job of the one it replaced. This is all transparent to the operator. Not all levels need be present, indeed, one or more may be omitted by operator choice. This is also handled in the software.

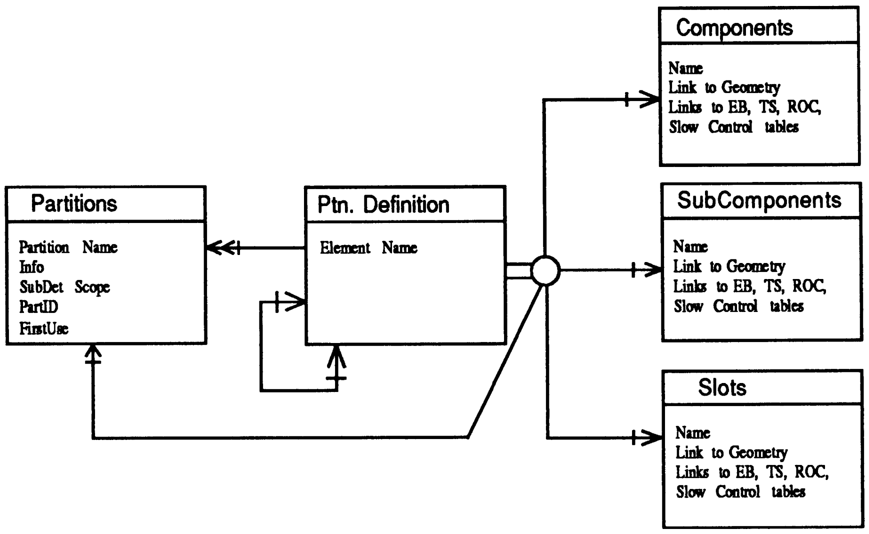

Figure 3: Partition Description ER Diagram

The Fastbus description (fig.4) [6] contains all segments and their connectivity, as well as all the device parameters (position, serial number, etc.). This reflects the state of Fastbus at any time, and can be checked with the hardware layout. The Slow Control part describes the sub-system used to control voltages, currents and temperatures for the detector and electronics.

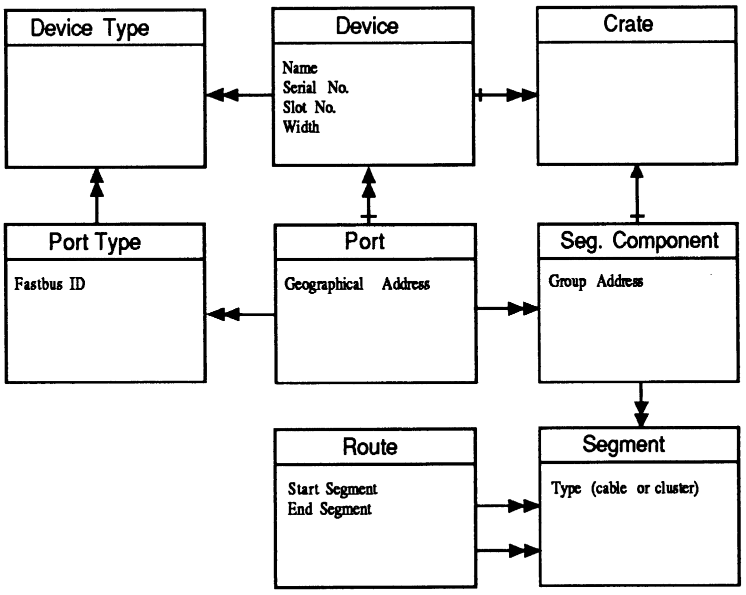

Figure 4: Subset of Fastbus Description ER Diagram

3. Implementation Architecture

The DAQ is an online, real-time system. Overall performance is highly dependent on efficient database access, so this influenced the implementation greatly. Commercial software (Oracle) had too large an overhead for simple access, but was used for backup and manipulation of the database structure. This provided preservation of existing data when new tables or relationships were added.

The programmer sees the tables as Vax Fortran structures, giving readable code and good performance. Tools to generate the necessary include files from the Data Dictionary, list the table contents, generate Oracle tables, move the data to and from Oracle and monitor usage were produced. Packages to access and modify the data were written using a general form shown in fig.5. This ensured the update protocol was obeyed and hid structural changes from the user [7]. To supplement the user-callable routines, dedicated database editors provide a consistent operator interface. They give access to the data (via menus) and the option to perform actions.

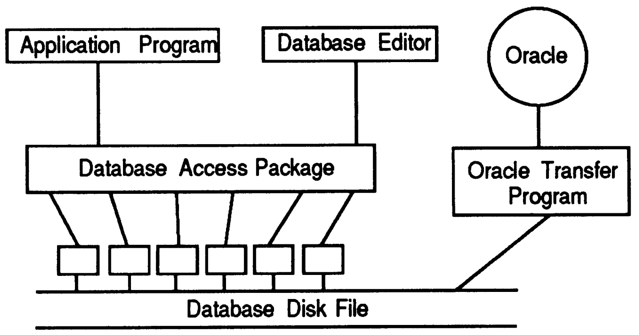

Figure 5: General Database Organisation

Sharing the data within one Vax was achieved by using global sections. This allows a section of memory to be shared between processes. Any update is immediately seen by all processes accessing this memory. Permanent storage was on disk; the global section is *mapped* to a file and the operating system updates the file on request. However, the DAQ system is distributed between a group of VAXes in the form of a cluster. In order to maintain database consistency between machines, a method for communicating changes between nodes was devised, using the locking mechanism supported by the VAX operating system. Any changes made to a database were therefore made in an interlocked fashion, using a common package to perform the access control. Write access is requested, the data modified, a disk update performed and an update message sent to any process (on all VAXes) with read access to the database. Any process reading then retrieves the latest copy of the data from the disk file (see fig.6).

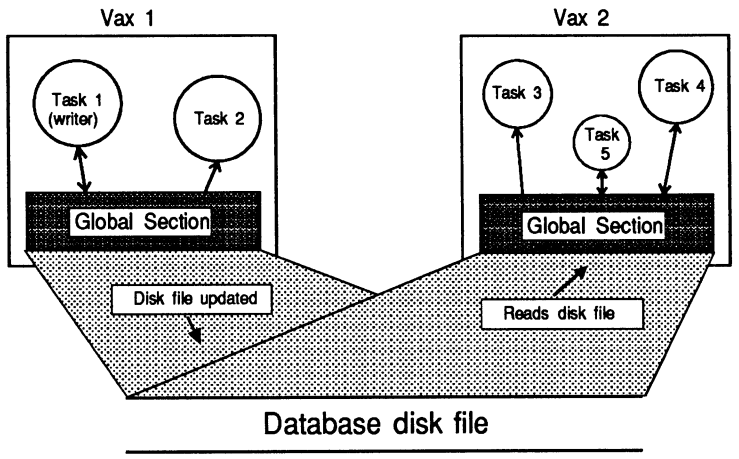

Figure 6: Cluster-wide database updating

This method produces no overhead for read access, which was the primary criterion, as no check need be made to see if data has changed. When an update is made, any process accessing the database receives an interrupt which may take up to one second to complete (this is a function of file size). The modifying process must wait for this to finish before continuing.

4. Current status

The system has been in constant use since ALEPH began taking data. The flexibility was exercised when overcoming a number of hardware failures, including the loss of the main VAX at one point. A new sub-detector has been successfully added, requiring no code changes (other than minor bug-fixes). Extra modules were added, requiring updates in the readout description and trigger connectivity. The Fastbus database supports continuous change as devices are added, removed, fixed and extended. Database usage has been rationalised to the extent that the exceptions to the presented scheme are minimal.

5. Conclusions

A flexible system must have a description from which all programs are driven. Complex databases are a necessary part of such a system, but a distributed database facility with adequate performance and the required functionality is not available commercially. The exploitation of the facilities of the VMS operating system have allowed these problems to be overcome in-house.

Since the detector, readout and fastbus are described in databases, the system could be used with another experiment. The Fastbus database in particular can describe any interconnection of segments that satisfies the Standard.

One enhancement would be the extension of the readout database to describe the front-end electronics. This was felt too sub-detector specific. A portable implementation of the databases would provide extra flexibility, but performance may always be the major constraint.

During the system evolution, certain Software Engineering techniques and principles were used more than others. The most useful stage was during the design period. The initial design remained essentially frozen. Firstly because the design did not require changes in many cases, but also because the tools available did not make a re-design easy. Propagating changes to the code had to be done manually. However, a widespread understanding of good coding techniques (modularity, information-hiding, well-defined interfaces etc.) prevailed throughout the group. This, it was felt, contributed greatly to the robustness and consistency of the final system. Documentation was a problem area — the need for automation (perhaps via interface specifications *à la* ADA ?) became very apparent. Software updates became coherent following the introduction of disciplined version releases.

6. Acknowledgements

Much of the inital Fastbus Database design was done in collaboration with CERN/DD division and the DELPHI experiment. Many of our colleagues have developed sub-detector databases using this method and we thank them for their contributions.

References

1. ALEPH: A Detector for Electron-Positron Annihilations at LEP, ALEPH Collaboration. CERN-EP/90-25 (Submitted to Nucl.Inst.Methods)
2. Requirements for the Aleph DAQ: ALEPH NOTE 115 DATACQ 84-1
3. Use of Software Engineering Techniques in the Design of the Aleph Data Acquisition System, T.Charity, R.McClatchey, J.Harvey: Computer Physics Communications 45 (1987) 433-441
4. ALEPH NOTE: ADAMO Notes V3.0, R.Brazioli, S.M.Fisher, P.Palazzi, W.R.Zhao
5. DAQ Software Architecture for ALEPH, A.Belk *et al.* Proceedings of the Real-Time Conference, Williamsburg, Virginia, May 1989
6. Fastbus System Management: E.M.Rimmer, DD Division, CERN 1988
7. The Aleph Online Database, A.Belk, D.R.Botterill, J.Harvey: ALEPH 88-205 DATACQ 88-38 Nov.1988

Note: All ALEPH notes can be obtained from the ALEPH Secretariat, CERN/EP, CH-1211 Geneva 23, Switzerland, upon quoting the reference number, title and author.