ALEPH 95-123
DATACQ 95-002
October 30,1995

EXPERIENCE USING DATABASES IN THE ALEPH DATA ACQUISITION AND CONTROL SYSTEM[*]

P. MATO, O. CARMONA, M. FRANK, J. HARVEY, B. JOST, D. MICHAUD, W.TEJESSY
CERN, 1211 Geneve 23, Switzerland

A. PACHECO
Institut de Fisica d'Altes Energies, 08193 Bellaterra (Barcelona), Spain

O. CALLOT, I. VIDEAU
Laboratoire de l'Accelerateur Lineaire, 91405 Orsay, France

D. BOTTERILL
Rutherford Appleton Laboratory, Oxon OX11 OQX, United Kingdom

The Aleph data acquisition and control system (DAQ) relies heavily on the use of databases to hold descriptions of the detector, the readout, trigger, control and data monitoring systems. In this paper we review the overall architecture of these databases and to describe the problems we have encountered with the current system mainly in the area of the physical implementation and standardization which influence directly the effort needed for long term maintenance of the DAQ system for a big experiment.

1 Introduction

The DAQ system for the Aleph experiment [1] has been running successfully during the last 5 years and needs to be maintained for another 5 years. Just to maintain such a system as it is, is a challenge in itself, especially taking into account the manpower available. In order to improve the maintenance, the reliability and the operation of the system we had to re-engineer parts of it to simplify and standardize as much as possible. All the modifications and upgrades since the startup in 1989 have been with these objectives in mind. One of the areas, where we could improve its maintenance and enhance its operational aspects, is the area of databases.

Our management and configuration software [2] relies heavily on the use of databases to hold descriptions and parameters of the detector, the readout, trigger, control and data monitoring systems. It is important that the contents of these databases are described in a complete and consistent way and without redundancy. Using these databases makes our DAQ system completely data-driven. This gives us a good deal of flexibility since changes on the configuration or running conditions do not require any program to be re-compiled or re-linked, it only demands a few changes in some of the

[*] Presented at the *Computing in High Energy Physics* conference, Rio de Janeiro, September 18-22, 1995

databases. Also, the fact that the parameters are stored in a database insures that all the programs use the same set of parameters, thus the coherence is guaranteed across the system. And finally, data-driven programs can be re-used in different environments and running conditions, thus reducing the maintenance load.

We have currently about 40 databases, each one specialized to a given domain of the on-line system. Examples of these databases include the Fastbus database, containing descriptions of the front-end electronics, the VME database, which describes the readout configuration, the trigger and slow control databases, as well as databases describing software components, such as the histogram database.

2 Main features for the current database design

2.1 Many relatively small databases

If one looks at the current implementation of our DAQ system, one soon realizes that are being used many rather small databases instead of using a big and unique database, as is the case for example in the off-line data analysis and reconstruction system. In fact, the exact number of databases depends on how we define them, it is of the order of 40. The reason for this diversity of databases is that the whole data is partitioned into small ones containing closely related information (domains). The information is partitioned following these two axes:

- Functional decomposition of the complete system in common subsystems and functional units. For example we have databases for task scheduling, detector description, read-out description, Fastbus configuration, data monitoring, slow control, etc..
- Sub-detector specific databases which are needed since each sub-detector uses different detection techniques and requires different nature and number of parameters to be stored. This decomposition also allowed several groups of people that built the sub-detector and developed the software to work in parallel. Examples of these databases are the calibration constants, geometry, running parameters.

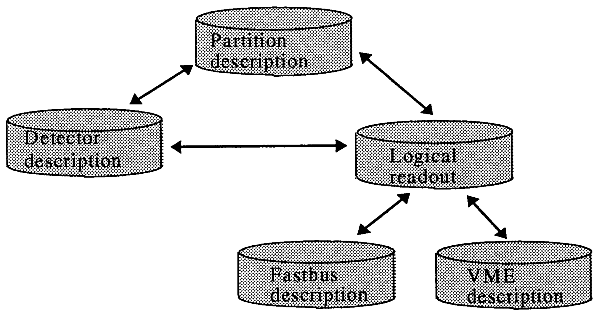

Figure 1: Some of the databases for the central read-out system and their relationships.

The domain specific databases are related between them as is shown in Fig. [1] as an example for some of the central read-out databases. The complexity of these databases in terms of number of tables varies from 1 to 20. And the sizes from few Kbytes to several Mbytes.

2.2 Entity relationship modeling

Most the databases, specially the central ones which are in common to all the sub-detectors has been designed using the entity relationship data model [3], which results in a tabular structure of data with relationships. The database schema is described using a Data Dictionary Language (DDL).

2.3 Distributed databases and concurrent access

Each task from the more than 100 which run on the system needs to access a few databases to gather all the necessary parameters. These databases are distributed among several nodes in the on-line cluster. On some occasions, like the start of a data taking run, many of these tasks need to access the database at the same time.

2.4 Physical implementation

The majority of databases are physically implemented as disk based VMS global sections (shared memory). Oracle is used as a backend for backups and management. The Fortran and C data structures are generated automatically from the DDL. However, the developer must write a set of access routines and an editor for each database. Other physical implementations co-exist like text files, indexed files, etc..

3 Problems and limitations

Based on our experience running and maintaining the system we have identified some of the areas where there are some problems or we are somehow limited. Here are some of them:

- Many domain-specific databases make the system more modular so in some way more maintainable. However it complicates the implementation of the relationships which exists between domains. The way we have overcome this complication is by duplicating small portions of information or by using implicit links based often on strong naming conventions.
- Due to the fact that different people collaborated in the development, you get different application program interfaces (API) to the data and editors with different look-and-feel. And since similar code had been produced many times you may get similar bugs replicated. This diversity discourages even more the navigation between databases and complicates the interaction to databases for the people operating the system.
- The current physical implementation renders difficult the evolution of the system since changes in the database schema are not always easy.

4 Areas of improvement

4.1 Single database schema

Conceptually, we should view all the databases as a single database schema. The links between databases should be made explicit, so that duplication of information is completely avoided thus eliminating potential incoherences. See figure 2.

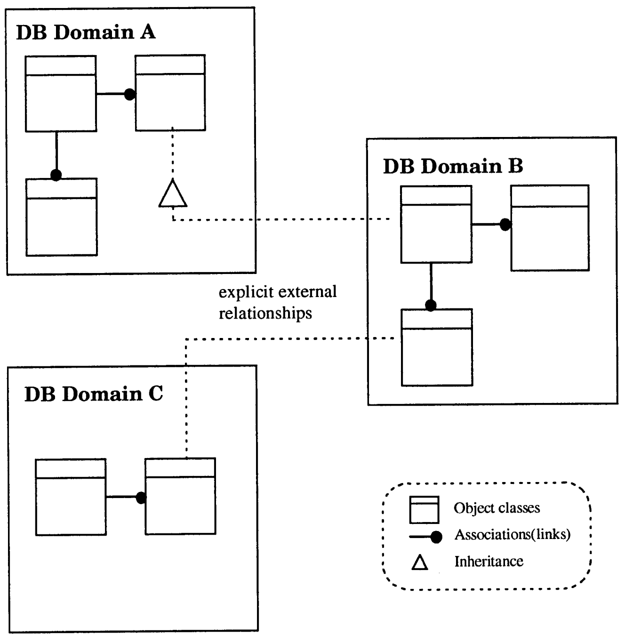

Figure 2: Sketch showing the idea of domain-specific databases with external relationships.

4.2 Data access routines

The application code should never access the data directly. We think that at least two layers of software should be in between. The first layer should be provided by a generic database management system (DBMS) and the second layer should be domain specific and implement only the queries and manipulations which are needed by the application. The user application should only call routines on the top layer. With this layer model the evolution and the eventual replacement of the DBMS should be transparent.

4.3 DBMS Standardization

The benefits of using a unique DBMS which satisfies our requirements are numerous. It does not need to be the state-of-the-art industry standard but at least it must be the standard inside the collaboration. In particular, one should standardize the functionality of the application program interface.

5 DBtool

We have recently developed a simple database framework called DBtool [4] to be our standard DBMS. DBtool should overcome all the limitations we have encountered with the old database implementation.

5.1 Some features of DBtool

DBtool supports database external links by the mechanism of importing remote classes. Using this feature we can build the global schema maintaining the idea of separated database domains. External and internal links are viewed the same way through the application program interface. Thus, for the developer it is very easy to navigate the entire global database.

DBtool stores the database schema as metadata in the database itself, thus allowing us the possibility of creating data-driven generic tools. In particular, with DBtool we have developed a generic editor which is used to fill and update all databases. This is the way we provide a uniform user interface to the people operating the DAQ system.

DBtool provides program interfaces to Fortran/C/C++ and it runs currently on VMS, OS9 and UNIX. The performance is improved by using disk caches and hashing algorithms for data retrieval and is comparable to the performance achieved using the global section implementation.

5.2 Current Usage

Since the beginning of this year all new databases have been implemented using DBtool. Also, a number of the old databases have been converted to use the new tool. In some of them, the data model has been enhanced, on others each entity set has been translated to a DBtool class. These new developments and re-implementations have allowed us to consolidate the tool and verify that no functionality is missing.

We intend to continue converting databases and to review the global schema to make use of the new features provided, such as inheritance and external links. In particular, inheritance can be very useful for sub-detector databases which contain specific information and which then can be added in a very natural way to common objects.

6 Conclusions

There are many benefits resulting from developing a data-driven DAQ system such as are flexibility, coherence, code re-usability, ease of maintenance, etc.. We believe that is worth the effort even if it costs more to produce. One way to encourage developers is to provide an easy to learn and use DBMS.

The standardization of the application program interface to the DBMS is needed from the beginning of the development. The functionality of it should be fixed. However, the calling sequence, language binding, the actual DBMS can change and evolve during the lifetime of the experiment.

We believe that converting our databases to DBtool and revising the global schema will improve the operation and maintenance of our system.

7 References

1. W. von Ruden, *IEEE Trans. on Nucl. Sci.,* vol **36-5,** 1444-1448 (1989).
2. A. Belk et al., *IEEE Trans. on Nucl. Sci.,* vol **36-5,** 1534-1540 (1989).
3. P.P. Chen, *ACM Trans. on Database Systems*, vol **1-1**, 9-36 (1976).
4. P. Mato, D. Michaud, *Dbtool user's Guide*, Aleph Online internal note.