[REDACTED]

CERN LIBRARIES, GENEVA

CM-P00065564

CERN-CN 90-7
C2

CERN-COMPUTING & NETWORKS DIVISION
CN/90/7
April 1990

**Databases for Large Detector Systems -
Experiences at LEP and Future needs**

E.M. Rimmer

(Invited Talk presented at the Computing in HEP Conference
Santa Fe, New Mexico, USA, April 1990)

DATABASES FOR LARGE DETECTOR SYSTEMS – EXPERIENCES AT LEP AND FUTURE NEEDS

E.M.Rimmer
CERN, 1211 GENEVE 23, Switzerland

ABSTRACT

The physical size of the LEP detectors at CERN, the quantity of information which they generate, and even the number of collaborators needed to build and run them, has shifted a large fraction of the associated software problems into the area of data management. Managing large volumes of data of all kinds for a variety of applications, from transport and storage to retrieval and analysis, has become, and will remain, the primary concern of many HEP programmers. All four LEP experiments have used a mixture of commercial and home-grown database management systems to exploit their detectors. Despite some efforts to achieve coherence the same data appear in different guises in different systems, and essentially equivalent code has been written several times over to access and manipulate these data. What can we learn from the LEP experience to ensure that, for detectors at the LHC and SSC, we overcome the difficulties inherent in the information management challenge which they present?

THE SCALE, DURATION AND COST OF THE PROBLEM

Electron-positron bunches cross in LEP at 3×10^4 Hz. Interesting Z^0 events occur at about 1 Hz and contain $10^7 - 10^8$ bytes of raw data. Proton bunches will cross in the LHC (if it is built) and the SSC at around 6×10^7 Hz. Inelastic events will occur at about 10^8 Hz; W and Z^0 events at around 10^2 Hz; Higgs events at perhaps 0.1 Hz. These events will also contain about 10^8 bytes of raw data [1]. Present storage tapes can be recorded at a sustained rate of more than 10^5 bytes/second. Each LEP experiment is expected to accumulate some 10^7 events. Together with data summary tapes and simulated data, these will fill around 10^4 cartridges per experiment [2]. We can expect at least the same storage rates and volumes at the future colliders.

The detector read-out technology, prompt triggering and event filtering requirements at an LHC/SSC experiment are therefore vastly different from those at LEP. The number of processing elements required and the problems of correlating data and selection decisions flowing through many parallel pipelines will need a level of control, monitoring and organisation far beyond any encountered at LEP. Further down the data handling 'funnel', things begin to be more recognisable, but we are still faced with a data management load somewhere between 10 and 100 times heavier than at LEP.

The expected lifetime of a LEP/LHC/SSC experiment is far greater than the participation period of most collaborators. Cost-effective in-house software must survive changes in personnel as well as detector upgrades and variations in the generated data. It must also survive changes in hardware platforms and technologies, and partial replacement as cheaper and more effective commercial products become available. The real software costs of any large project were already known to have overtaken hardware costs in the 1960s, with maintenance absorbing around 70% of the total budget. An analysis of the maintenance activities of 487 different installations, made in 1979 [3], showed the top time/money consumers to be:

- 1) almost 42% for code modification to follow the evolution of the originally perceived requirements;
- 2) over 17% for code modification because of changes in data formats;
- 3) over 12% for emergency fixes; 9% for post-delivery debugging.

We have little evidence that the situation has changed dramatically during the past decade, at least not in HEP. Item 1) indicates that software isn't as soft as it should be, while 3) shows that it isn't as correct as it should be, both implying the need for a more rigorous approach to software construction, a case which has been eloquently argued elsewhere [4]. However, item 2), data 'fixups', is due primarily to a mingling of management code, which handles data storage, representation and access, with manipulation code, which ought to be dealing solely with the information content of the data. This implies the need for a cleaner separation between these two views of data, by using programming techniques and languages which support this distinction [5] and by fully exploiting the increasingly powerful facilities offered by modern DataBase Management Systems DBMSs.

DATA MANAGEMENT FOR THE LEP DETECTORS

Physics data management at LEP has mainly involved handling the large volume of detector geometry and calibration data, and the associated raw and simulated event data. In what follows, we do not discuss the management of graphical data nor of large sets of histograms; suffice it to say that the situation there is not less diverse.

At LEP, a mixture of commercial DBMSs and home-grown data management packages, hereinafter referred to as data managers, is used for the exploitation of the detectors and analysis of the event data. They run on various platforms and use a variety of data structures which are not always trivially interchanged between applications requiring access to the same information. In some cases, data are duplicated in several places in different guises, giving rise to potential inconsistencies.

The LEP experiments have passed the initial phase of detector commissioning and data-taking most successfully. But what of the coming years? Are we in a data management muddle? Have we created an *ignotum per ignotius*, a solution which is more complicated than the problem. If so, what were the causes? Was current database technology inadequate for our needs? Did we fail to use programming methods and languages which closely matched our problems? Did we correctly locate the boundary between buy-in and do-it-yourself, and follow it as it shifted with time?

The LEP collaborations were well aware of the data management difficulties which they were facing. In 1983, a working group of the European Committee for Future Accelerators ECFA made a study of the use of databases in HEP and published a set of recommendations [6]. It is therefore informative to ask to what extent these recommendations have been adopted at LEP and if not, why not. To do this, we will group the 10 original recommendations into families.

USE OF COMMERCIAL DATABASES

"There would be many advantages in using commercially available DBMSs in HEP...flexibility and ease of use clearly point to the need for a relational database...there is an immediate need for the major HEP computing centres, especially CERN, to make suitable RDBMSs available to users..."

As we shall see, these recommendations were more or less followed, but their impact was somewhat limited. During LEP detector design and commissioning, commercial databases were still primarily aimed at relatively simple bulk data storage and retrieval. They were expensive and ran only on powerful platforms. Used interactively or in non-time-critical batch jobs, they were ideal for bookkeeping applications such as stock inventories and personnel records. CERN support for the ORACLE RDBMS [7] was set-up in the mid-1980s. Initially installed on the central IBM and VAX computers, ORACLE is now available on workstations such as Suns and Apollos and personal computers such as Macintoshes and IBM PCs. In 1987, the installation of SQL*Net [8] for remote access to central ORACLE databases greatly increased the use of these systems. ORACLE was employed extensively in building the LEP machine [9] and is now the basis of almost all administrative information handling, both for on-site and distributed administration of the LEP experiments and for the running of the laboratory [10].

CERN negotiated a substantial discount on ORACLE products installed at institutes collaborating in the physics program, but this offer did not arouse much interest and was withdrawn at the beginning of 1990. External institutes are mainly involved in detector development and data analysis, activities which make heavy use of large sets of detector geometry and calibration data. With the exception of ALEPH, the LEP experiments did not use ORACLE for these physics-oriented applications.

Many ORACLE users complain about its old-fashioned menu-style interactive interface SQL*Forms. The Macintosh-based products Hypercard and 4th Dimension provide alternative interfaces, at least for Mac users; both are used by OPAL [11]. It seems that 4-D will attract an increasing number of users, particularly as it can now be linked to a remote ORACLE DBMS via SQL*Net, a 'best of both worlds' solution.

HOME-GROWN DATA MANAGERS

"Simple HEP database packages will continue to be needed...if a much more sophisticated system is needed, then a DBMS should be used...users developing higher level software of a general nature [...] should be urged to write as much as possible in the form of add-on packages which can be used by other groups...before the development of very sophisticated or complicated packages is undertaken for a single experiment, careful consideration should be given as to whether the advantages to be obtained will justify the work involved..."

Home-grown data managers are employed for memory, data and file management [12]. They handle mixed data-type records and dynamic memory allocation, features which are absent from FORTRAN, the traditional HEP programming language. The data structures supported are mainly hierarchical trees, with up/down/next/previous links, connecting for example, events with (down) tracks with (down) points. These links optimise the efficiency of searching and navigating through the structures. The packages also have simple file management facilities for data exchange between hosts and external direct access devices, and for data exchange between host computers with different internal data representations. To simplify application programs, the low level data and file access routines are usually supplemented by higher level procedures.

These data managers form a natural part of the off-line analysis environment, where they are used to manipulate detector geometry and calibration data, and raw and simulated event data. During the past few years, they have also been incorporated into the on-line environment, for event selection in embedded processors or farms, and for sample analysis in workstations linked to the data collection computers. Two data managers are used by the

LEP experiments; DELPHI, L3 and OPAL use ZEBRA [13]; ALEPH uses the Bank Organisation System BOS [14].

The reasons given for using home-grown packages is that they are portable, efficient and free of charge. A non-technical reason, which is still valid, is that some products, in this case ORACLE, cannot be shipped to all countries in the collaboration. Recent political developments may ease this situation. It should be noted that even these kernel systems become unacceptably inefficient as they fill with large amounts of data. To what extent they can be said to be free-of-charge depends on your accounting methodology! Another argument is that many of the facilities offered by a commercial DBMS are luxuries which are purchased but not used. However, all of the LEP experiments have eventually been obliged to implement mechanisms to handle data integrity, concurrent access and transaction processing, that is, 'roll-back' if a related series of operations cannot be completed satisfactorily.

One serious HEP requirement is the need to stamp records with a creation time and period of validity, to allow re-creation of run and detector status in the past, a necessary feature of physics analysis, which may be performed long after data collection. This mechanism is not trivially implemented directly in a relational DBMS. Also since analysis is done at several different institutes, it is desirable to distribute small sets of update records, sometimes at quite frequent intervals, rather than an entire export of a central database. This might be a relatively simple add-on to a commercial DBMS if the history-archiving problem was solved. What we are really looking for is a commercial system which can be used as a secure back-up store and master data repository, and which is cleanly and adequately interfaced to a portable, efficient HEP data and file manager. If greater use were made of languages with built-in complex data structures and memory management, need for add-on features would be correspondingly reduced.

ADDITIONAL LEP PACKAGES

The ALEPH collaboration made a concerted effort to adopt a coherent data view by using the Entity-Relationship Data Model [15] as the basis for all data descriptions, from detector geometry to collaborators' home and e-mail addresses. The modelled data are stored in tables which are created, filled and accessed using ADAMO software [16]. ADAMO provides a wide range of facilities, from a Conceptual Schema Processor to a Table Access Package. ADAMO data structures are mapped by callable interfaces onto ORACLE tables, BOS banks, FORTRAN COMMON blocks and VAX-FORTRAN RECORDs, depending on the application.

The ALEPH detector description is held in ORACLE; event data are held in BOS formats. The detector DB is used for both off-line and on-line applications. Data entry to this DB is performed via Structured Query Language SQL commands [17]; data retrieval is via ADAMO with BOS mapping. The interactive analysis and reconstruction programs operate entirely in the ADAMO/BOS environment and do not communicate with ORACLE. A Table Interaction and Plotting package couples ADAMO to the Physics Analysis Workstation PAW software suite [18]. The ALEPH online database is also held in ORACLE and is used for run control and status recording. Data entry and retrieval are performed via ADAMO with VAX/FORTRAN RECORD mapping, either in purpose-written programs or in association with an interactive menu package. Modifications to the online database are written back to ORACLE.

ALEPH personnel inventory data are also kept in a central ORACLE database. In this case, data entry is performed using interactive ORACLE SQL*Forms and data retrieval is done via ADAMO with FORTRAN COMMON block mapping [19]. These portable data

blocks are shipped to the collaborating institutes for local use in FORTRAN programs using the ADAMO Table Access Package. This facility, NADIR, which is also used by DELPHI, offers automated electronic mailing as well as the more usual directory services.

The DELPHI detector description is held in CARGO [20], a home-grown database system which has its own internal hierarchical structure and which uses a modified version of the Keyed Access Package KAPACK [21] to implement the access routines. Event data are held in ZEBRA formats. In the on-line environment, access to CARGO is achieved through a unique Remote Procedure Call server, which provides protection against concurrent access by queueing client requests. Data entry is performed by reading pre-prepared ASCII files into CARGO, or via a custom-built simple interactive package. In the interactive analysis environment, a buffer processor TANAGRA is used as the coherence factor inter-linking the various data structures [22].

The OPAL and L3 detector descriptions and event data are held in ZEBRA formats. Both experiments have implemented a data management system on top of the ZEBRA direct file access routines RZ, for use in the analysis phase, whether on- or off-line. The L3 package is DBL3 [23]; the OPAL package is OPCAL [24].

The ALEPH collaborators have benefitted from having some degree of uniformity in their data view, however they have not found it easy (maybe not even necessary) to by-pass some of the intervening data structure and access layers, say between BOS and ORACLE, nor to replace some of the ADAMO facilities by equivalent features of ORACLE. Managing ADAMO data structures in an underlying ORACLE RDBMS is complicated by the ADAMO implementation of inter-table relationships. Usually these relationships are established via hidden multi-table 'joins' at access time, using symbolic or numeric values to define conditions on the attributes which confer the relationship. For example,

```
SELECT name, homophone, instphone FROM collaborator, institute
WHERE collaborator.speciality = 'FASTBUS'
AND collaborator.institute = institute.name
```

ADAMO inter-table relationships however are implemented explicitly via a table column which contains the integer value of the number of the related row in the 'partner' table. These absolute pointers, which must be preserved in the mapped ORACLE tables, increase the efficiency of data access in memory but require special precautions when rows are inserted or deleted [25]. It is not possible to exploit relationships unless they have been foreseen and not easy to understand the 'reason' behind a relationship by simple inspection. All HEP data managers have linking pointers which are an intrinsic part of the associated data structures, blurring the distinction between the management and information content of the structures.

We can conclude that the recommendations concerning home-grown data managers were not really followed. A great deal of real effort was expended several times over to produce essentially equivalent code to examine, access and manipulate data, and to transform their formats between various producer and consumer processes. A fuller understanding of why this was necessary and why the effort was not concentrated in a single project might lead to an improved situation at LHC/SSC.

TAPE MANAGEMENT

"A greater degree of automation in the management of tape data would be desirable..."

Real progress was made in 1989 with the introduction of the Distributed File and Tape Management System FATMEN [26]. The requirements for FATMEN were compiled by

the LEP experiments plus UA1 and UA2, in collaboration with the CERN Data Division [27]. The experiments are in the process of converting to the new facility. FATMEN is a networked system which supports access to stored data in a consistent manner, regardless of the host operating system or the location or type of storage medium. FATMEN offers bookkeeping facilities for file storage and retrieval for physics data analysis e.g. the management of cartridge robots and disk farms. It has a command line interface and a FORTRAN callable interface, and has been installed on almost all of the HEP mainframes and workstations.

FATMEN uses the ZEBRA package to provide memory and data structure management and file exchange. The ZEBRA RZ direct file access routines are used to manage catalogue information stored in a hierarchical direct access directory structure. File identifiers are UNIX look-alikes, for example, //CERN/DELPHI/RAWDATA/RUN20. A ZEBRA server, resident at the production centre, provides remote clients with access to the FATMEN File Catalogue, maintained under ORACLE; at CERN the IBM CERNVM ORACLE RDBMs is used. A combination of ZEBRA file access facilities, local journal files, ORACLE journaling and roll-back is used to assure reliability of the catalogue, which is eventually expected to contain references to hundreds of thousands of datasets.

OPTIMAL USE OF DATABASES

"Greater awareness is needed within the HEP community of what DBMSs offer... pilot projects should be set up so that some experience with them can be obtained as soon as possible... standardisation at the SQL interface level is suggested both for interactive terminal use and embedded in FORTRAN programs..."

The Fastbus System Manager FSM [28] is an example of a project which aimed to fulfill these recommendations. The purpose of FSM is to provide both expert and non-expert users with a means to describe, verify and initialise any Fastbus read-out system [29], particularly a large and complex system, and to assist with diagnosis by tagging malfunctions at run-time. A serious attempt was made to gain some experience in modern software engineering methods and to exploit all the useful features of a commercial database [30]. Structured Analysis/Structured Design SASD techniques [31] were used to define the overall FSM software. Data Flow Diagrams provided a diagrammatic analysis of the problem and its solution, in terms of data and data transformation processes. Entity-Relationship data modelling defined the necessary set of Fastbus data and related database tables. A set of Structure Charts detailed the data transformation algorithms.

As an aside, it should be mentioned that the FSM team did not use any Computer Aided Software Engineering CASE tools, mainly because a coherent tool kit was not available at the time. This certainly reduced the potential efficacy of the methods, particularly in maintaining consistency between documentation and code. Nevertheless, the discipline inherent in the approach, and the enhanced understanding of the problem, made it possible to find a structured and modular solution which greatly simplified implementation and maintenance.

Having modelled the data, a relational database (ORACLE) was judged to be completely adequate to catalogue the hardware components of the system, to describe its physical connectivity and to define its physics data-taking functionality. The hierarchical Fastbus broadcast tree was trivially implemented by 'child of' up-pointers. The database essentially contains four sets of tabulated data:

- 1) *specification data* which define the physical characteristics of hardware types, such as the slot width of particular device type; these data are 'universal' and not experiment-specific;
- 2) *layout data* which define the location of equipment in a particular experiment, such as the slot address of a device in a crate;
- 3) *configuration data* which define which hardware is to be initialised/verified, and the physics data-taking functionality, such as Fastbus transaction routing requirements;
- 4) *system data*, generated by FSM from the first three sets of tables, which define the run-time contents of all programmable Control and Status registers and sequencing information which allows any fully-connected Fastbus system to be initialised.

The primary data tables, 1, 2 and 3, are defined entirely from an end-user viewpoint. They contain only descriptive information defining the components of the read-out system, how they have been put together, and what they should be used for. The navigational and searching methods of the RDBMS itself are used for access. The database tables are created using a standard FSM SQL command file. They are filled and inspected via a set of interactive SQL*Forms. All data management, such as importing and exporting data sets, is done using the facilities provided by the RDBMS.

FSM itself performs two separate functions, generating an operational view of the read-out system from the primary data, and interpreting this view at run-time. The FSM generator is run interactively via SQL*Forms. Data are read from the database into memory where they are used to produce coherent set of run-time parameters (programmable register contents) for any set or sub-set of the equipment. These parameters are combined with hardware connectivity and access sequencing information to produce portable output ASCII text files, which describe the procedures necessary for the initialisation and verification of the hardware described. These files are read from local store at run-time by the FSM interpreter, using a simple callable interface to application code. During interpretation, an FSM library of grouped Fastbus operations is used for performing specific initialisation sub-sequences, such as loading and enabling a segment interconnect. This library uses standard DOE/ER-0367 Fastbus subroutines for bus access [32]; these are available for all of the commonly used HEP host/Fastbus interface combinations.

When database or text file data are read into memory, appropriate temporary structure pointers are added to optimise access according the application. For example, parameters are stored in a balanced binary tree of records for efficient random access, whereas the initialisation sequence is stored as a simple linked list because only the *next* item in the sequence is needed at run-time. The database itself is not used at run-time, to avoid timing overheads often associated with database access, and to allow initialisation by a host processor which is not connected to the database. However, for small systems or in test environments, some groups, in particular L3, have found it convenient to link the generator stage directly with the interpreter stage.

At CERN, the FSM Fastbus databases are maintained in ORACLE running under VAX/VMS. The FSM generator is written in PASCAL with VAX extensions for string handling. PASCAL was chosen to provide mixed-type data records, dynamic memory allocation, and recursion to an unknown depth when handling the Fastbus broadcast tree. We also hoped that the code would be more readable and therefore more maintainable than C code! The current version of the FSM interpreter is also written in PASCAL and is presently only used in VAXes. The text files are in the form of C character arrays, and, if necessary, could be handled by an interpreter written in C running in an embedded processor, if such a processor did not have a PASCAL compiler or local storage. The interpreter has not yet been translated into C because of lack of demand.

FSM INTERFACE TO THE DBMS

The FSM interface to the DBMS is entirely through SQL-based facilities. Our experience has shown that standardisation at this level, following the ECFA recommendations, has paid handsome dividends. SQL*Forms provide a secure, multi-user interactive interface to the database. The FSM generator accesses the database via a set of generic routines, *find_unique_value*, *open_table*, *get_next_value*, *close_table*. At CERN, these routines are implemented using the ORACLE Pro-PASCAL pre-processor, which expands embedded SQL statements into PASCAL code. By the simple expedient of replacing the underlying database library, FSM has been ported with ease to an INGRES RDBMS [33] by the MEGA experiment at LAMPF [34].

The ORACLE SQL*Report generator has been used by DELPHI to produce textual data records in a customised format for interpretation by application code which was written before FSM. Prior to the existence of the database, these records had been manually (and carefully!) produced using a text editor.

DATA INTEGRITY CHECKING

When preparing software which is to detect hardware failures, other sources of error must be removed as early as possible in the chain between defining the primary data and accessing the physical equipment. A non-expert user should not be able to introduce errors because of ignorance. FSM exploited the DBMS itself to assist in this task. Some of the mandatory constraints of the Fastbus Standard are enforced as defined properties of the database tables, such as the uniqueness of a Fastbus PortType Identifier. Inter-table relations, for example that a segment interconnect must have ports on two different segments, are checked during interactive input using pre- or post-change SQL program triggers. Fields in the SQL*Forms windows correspond to columns in the database tables. The triggers are activation mechanisms connected to these fields, which allow SQL programs to be run before or after the datum in a given field is changed. These programs can issue pre-change advice such as 'if you insert a value here, you will have to insert another elsewhere', or post-change warnings, such as 'now that you have changed this, the database is inconsistent unless you change that too'. These checks serve as pre-condition assertions on data used by any database consumer, greatly reducing the amount of checking needed in the consumer code itself.

More complex Fastbus system constraints, for example that addressing routes must not contain loops, can only be checked by the FSM generator. These checks forestall inadvertent attempts to use an erroneous Fastbus architecture. A thorough understanding of the rules of Fastbus, partly encapsulated in the database proper and partly in FSM code, allows us to generate initialisation and parametric information which is known to be correct. Therefore, errors at run-time can only be due to hardware malfunctions or to a mismatch between the primary data provided by the user and reality.

Run-time verification consists of reading slave Fastbus identifiers to ascertain that the database accurately describes the physical system and that the hardware is basically functioning. Because the initialisation sequence is well-determined and has the characteristics of an increasingly deep probe, it is always known which equipment between the host processor and a particular segment or slave has been used without error. Therefore, malfunctions can be localised to one or a few system components, not necessarily a trivial achievement in systems such as Fastbus which can have complex connection topologies. Two pilot projects are underway to link FSM diagnostic output to rule-knowledge based Expert Systems, to provide the end-user with more detailed guidelines concerning hardware failures [35].

FSM is used by the LEP experiments DELPHI and L3 and by the MEGA experiment at LAMPF; a derivative version is used by ALEPH [36]. The DELPHI set-up is particularly demanding as it contains some 200 segments and almost 1000 connected Fastbus ports, with static and dynamic partitioning into sub-detectors and complex broadcast and transaction routing requirements for inter-partition communication. Because the ALEPH detector description is also held in ORACLE, they have been able to exploit the connection between detector components and read-out modules in their run-time system [37].

CONCLUDING REMARKS

Commercial DBMSs have gained a foothold in HEP, but are still regarded as unsuitable or inadequate for real physics applications. It is recognised that they provide reliable and secure data repositories, protection during concurrent access and integrity of multi-access transaction processing. However, they are criticised for their cost, lack of portability or availability, access time overheads and unfriendly human interfaces, amongst other things. Advances are being made on almost all of these fronts. Areas of progress in database technology which are of special interest to HEP include networked access, distributed storage and object-orientation [38]. Friendly front-ends networked to powerful central facilities will attract more converts to databases. The advent of interactive analysis systems such as Reason [39] might even herald a drift if not a full-scale flight away from traditional programming methods and languages, thereby reducing the range of facilities demanded of home-grown data managers. Reason, implemented in Objective-C on the NeXT computer, has a Macintosh-like human interface, and some large HEP FORTRAN packages such as LUND and MINUIT have already been incorporated.

Experience at LEP has shown that in fact *"there are many advantages in using commercially available DBMSs in HEP"*, however it still seems likely that *"simple HEP database packages will continue to be needed"*. Maybe an attempt can be made to marry these apparently opposed observations. It is certainly worth modelling the information management requirements for an LHC/SSC experiment. We should identify the entire set of data and establish an overview of the application programs which will access these data at all stages of detector design and exploitation. The data storage and retrieval environments should be studied to understand whether different applications use separate data sets or views of the same sets. Special needs, such as remote access or fast access, should be clearly defined.

Once a design study has been completed, it will be possible to judge to what extent commercially supported packages or some features of them can be used. If it emerges that these packages provide approximately what is needed, we should set up joint projects with industry to improve the weaknesses and fill the remaining gaps. Recent experience at CERN and elsewhere indicates that joint projects are usually more successful than preparing specifications for packages which are entirely produced by outside software houses; this opinion has been expressed by both partners in such projects. The reason is partly because members of the HEP community have only limited experience in preparing technical specifications of the necessary quality and clarity, and partly because some requirements only become obvious once a project is underway, sometimes only during 'alpha-testing'. A joint project approach also eliminates some of the feelings of insecurity engendered by 'not invented here' packages, and ensures a certain familiarity with the 'internals' of these packages, which is necessary given the evolution and maintenance requirements of modern HEP software.

EPILOGUE

In case anyone believes that life becomes simple once everything is 'databased', we end with a sobering story. The CERN Data Division DD recently changed its name to Computing and Networks CN. Some units will move to the Electronics and Computing for Physics division ECP later in the year. The following quotation is taken verbatim from minutes of a recent meeting: *"We are having consistency problems. Not all databases have changed from 'DD' to 'CN'. The real problems are still ahead of us when we switch to a 3-letter name "*. LEP, LHC and SSC are all 3-letter names, but the real problems may still be ahead of us!

REFERENCES

- (1) J.Renner Hansen, Working Group Report on Triggering and Data Acquisition, Proc. Workshop on Physics at Future Accelerators, La Thuile, Italy and Geneva, Switzerland, CERN Yellow Report 87-07, Vol.1, pp 274-294 (1987)
- (2) D.O.Williams, Computing on the eve of LEP data taking: are we ready?, Proc. Computing in HEP Conf., Oxford, UK, pp 8-14 (North-Holland,Amsterdam,1989)
- (3) B.P.Lientz and E.B.Swanson, Software Maintenance: a User/Management Tug of War, Data Management, pp 26-30 (April 1979)
- (4) G.Kellner, Software Developments using Structured Techniques, Proc. VMEbus in Research Conf., Zürich, Switzerland, pp 21-33 (North-Holland,Amsterdam,1988)
- (5) P.F.Kunz, An Introduction to Object-Oriented Programming Techniques for HEP, paper presented at this conference
- (6) J.Hart, Databases and Bookkeeping for HEP Experiments, ECFA Working Group on Data Processing Standards, Report of Sub-Group 11, ECFA/83/78 (1983)
- (7) ORACLE Corporation, 20 Davis Drive, Belmont, California 94022, USA
- (8) F.Hemmer, SQL*NET at CERN, Proc. 7th European ORACLE Users Group Conf., Brussels, Belgium, pp 113-134 (1989)
- (9) J.Schinzel, Using ORACLE to build a machine 27 kilometers long, Proc. 7th European ORACLE Users Group Conf., Brussels, Belgium, pp 57-65 (1989)
- (10) G.Ferran, A.Koppanyi and A.M.Osborne, The Computer Centre Database Tables, User Registration, Interface Tools, CERN/DD/US/CCDB/1, (1990)
- (11) H.Burckhart, OPAL slow controls (Hypercard); P.Le Du, OPAL electronics inventory (4th Dimension); private communications
- (12) R.Mount, Database Systems for HEP Experiments, Proc. Computing in HEP Conf., Asilomar, Computer Physics Communications, Vol.45, Nos.1-3, pp 299-310, (North-Holland,Amsterdam,1987)
- (13) R.Brun and J.Zoll, ZEBRA User Guide, CERN Program Library Long Writup Q100 (1987)
- (14) V.Blobel, The BOS System, DESY Report R1-88-01, DESY, Hamburg (1988)
- (15) P.Chen, The Entity-Relationship Model - Towards a Unified View of Data, ACM Trans. on Database Systems, Vol.1, No.1, pp 9-39 (1976)
- (16) M.G.Green, ADAMO Data System, ALEPH Note 89-47, COMPRO 80-2 (1989)
- (17) Database Language SQL, ISO Standard 9075 (1987)
- (18) R.Brun, O.Couet, C.Vandoni and P.Zanarini, PAW, Physics Analysis Workstation, CERN Program Library Long Writup Q121 (1989)
- (19) F.Blin, S.M.Fisher, W.G.Moorhead, P.Palazzi and Z.Qian, NADIR, the New ALEPH Directory, ADAMO Application Note 10 (1986)

- (20) G.P.Gopal, F.Carena and V.Perevozchikov, CARGO, a Hierarchical Database Management System, DELPHI Note 86-28, PROG-46 (1986)
Y.Belokopytov, S.Gumenyuk, V.Perevozchikov and R.Yamaleev, Detector Description Application Package, DELPHI Note 88-87, PROG-121 (1989)
- (21) R.Matthews, KAPACK, Keyword Access Package, CERN Program Library Long Writeup Z303 (1983)
- (22) D.Bertrand, DELPHI Interactive Analysis and TANAGRA, Proc. of the Computing in HEP Conf., Asilomar, Computer Physics Communications, Vol.45, Nos.1 – 3, pp 207 – 213, (North-Holland,Amsterdam,1987)
- (23) S.Bannerjee, F.Bruyant, R.Mount and E.Nagy, DBL3, a Software Utility Package for the L3 Databases (1989), available from R.Mount, CERN/L3
- (24) R.Cranfield, B.Holl and R.W.L.Jones, OPAL Calibration Database OPCAL User Guide, OPAL Note OFFL/36/0003 (1989)
- (25) A.Belk, The ALEPH Online Detector Description Database, ALEPH Note 89-129, DATACQ 80-20 (1989), see section 1.2
- (26) J.Shiers, J.Boucrot, L.Palermo, V.Innocente and J.Goldberg, FATMEN File and Tape Management System, CERN Program Library Long Writeup Q123 (1989)
- (27) C.Curran et al, The FATMEN Report : Experimental Needs, CERN DD Report, DD/89/15 (1989)
- (28) P.M.Ferran, E.M.Rimmer and P.L.Saetre, Fastbus System Management User Guide, DD Division, CERN (1989), available from E.M.Rimmer, CERN
- (29) US NIM Committee, IEEE Standard FASTBUS Modular High-Speed Data Acquisition and Control System, ANSI/IEEE Std 960 (1986)
- (30) E.M.Rimmer, A Database for Fastbus, CERN School of Computing, Troia, Portugal, CERN/DD/87/23 (1987)
- (31) T.DeMarco, Structured Analysis and System Specification, Yourdon Press, New York (1978)
- (32) US NIM Committee, FASTBUS Standard Routines, US Department of Energy, Washington D.C.20545, USA (1987)
- (33) INGRES, Relational Technology Inc., 1080, Marina Village Parkway, Alameda, CA 94501, USA
- (34) T.Kozlowski, Los Alamos Meson Physics Facility, private communication
- (35) E.Trasatti, F.Corazzini, S.Falciano, L.Luminari and E.M.Rimmer, An Intelligent Tool for Fault Diagnosis in Fastbus Systems, and
F.Block, FB_EXPERT: An Expert System for the Management and Error Diagnosis of Fastbus Networks, Proc. Workshop on Software Engineering, AI & Expert Systems for HEP & Nucl. Phys., Lyon, France (March 1990); to be published
- (36) D.R.Botterill, Fastbus Database and Initialisation, ALEPH Note ALEPH 88-206, DATACQ 88-39 (1988)
- (37) A.Belk, D.R.Botterill and J.Harvey, ALEPH Online Database, ALEPH Note ALEPH 88-205, DATACQ 88-38 (1988)
- (38) M.L.Van Name and B.Catchings, Serving Up Data (Networked DBMSs), DataBase Trends, BYTE, pp 259 – 264 (September 1987)
R.Davis, Sharing the wealth (Distributed DBMSs), idem. pp 267 – 274
J.Dawson, A Family of Models (Object-Oriented DBMSs), idem. pp 277 – 286
- (39) P.F.Kunz, B.Atwood, R.Blankenbecler and B.Mours, Reason, Proc. Workshop on Software Engineering, AI & Expert Systems for HEP & Nucl.Phys., Lyon, France, March 1990; to be published