
a1996001a

DESY 95-236
December 1995

ISSN 0418-9833

The Physics Analysis Environment of the ZEUS Experiment

Lothar A.T. Bauerdick, Oleg Derugin[a], Dave Gilkinson,
Matthias Kasemann, Olaf Mańczak

Deutsches Elektronen Synchrotron, Notkestrasse 85, 22603 Hamburg, Germany

The ZEUS Experiment has over last three years developed its own model of the central computing environment for physics analysis. This model has been designed to provide ZEUS physicists with powerful and user friendly tools for data analysis as well as to be truly scalable and open.

1 Introduction

During summer 1992 the HERA collider at DESY delivered its first luminosity and the ZEUS detector recorded its first data. At that time the ZEUS computing environment consisted of some few dozens of workstations (DEC and SGI), a parallel reconstruction facility running on multiprocessor SGI machines and there was a strong commitment in favor of computing solutions based on inexpensive RISC/UNIX technology. Nevertheless, a large fraction of the physics analysis and software development was done on the IBM 3090 mainframe and on the central DESY VAX cluster. In addition, the IBM mainframe with its STK tape library acted as our central repository for experimental and Monte-Carlo data.

In this paper we describe the evolution and status of the physics analysis environment at ZEUS beginning from this initial state.

2 First Experience

During the 1992 data analysis period we have made three fundamental *"discoveries"* concerning efficient and effortless data analysis. First, we found that the process of analyzing data can be considered as an infinite loop consisting of two steps:

- *interactive analysis* - reading mail and news, writing papers, analyzing ntuples, plotting histograms, *thinking*, coding an idea into a program etc. (i.e. developing analysis software)
- *batch analysis* - running an analysis program on a large sample of experimental data to check the idea (this can take 5 minutes to 1 week), then back to the interactive analysis...

It is natural and obvious that workstations (or more generally workgroup-servers and X-terminals) with large graphical screens and powerful computing resources are extremely convenient tools for interactive analysis. These workstations, however, do not address and do not solve all the problems of batch analysis. In particular, the access to a huge (several TBytes) remote tape library or to large amounts of

[a]On leave of absence from Moscow State University, Institute of Nuclear Physics

data available within a distributed network filesystem is significantly limited by the network performance.
Thus, (this was our second *"discovery"*) the use of workstations does not eliminate the need for central *mainframe-like* computing facility with fast access to experimental data but, to the contrary, enhances that need.

Finally, we found that in an environment where about 50 physicists (this was our estimate in 1992) work simultaneously analyzing more or less the same data it is highly reasonable to keep a large sample of selected physics events on disk. The main goal was to be able to analyze the whole sample quickly and to use *(share)* disk resources efficiently. This approach prevents unnecessary data duplication by different users when the right tools are provided.

3 ZARAH – The Central Batch Facility

The outcome of our 1992 experience was a proposal for ZARAH *(Zentrale RechenAnlage für HERA Physik)* the central computing facility for the ZEUS experiment at DESY. Among our major short term goals were:

- rationalized and optimized *path to data* in order to avoid network bottlenecks,
- a powerful and user friendly batch system, able to run at any given time at least three jobs processing some 20 events/sec, with direct access to
- a fast disk store for a large sample of selected events.

Our long term goal was to gradually eliminate the IBM from the offline processing both as a computing facility and data server. The details and development of this system over time is discussed elsewhere in these proceedings [1,2,3], but we summarize the current configuration here:

- 600 MB of fast disk storage available, enough to store a complete year's worth of compressed MINI-DST data and selected events from previous years
- Two SGI Challenge XL multiprocessor machines (34 processors total) to serve as batch platforms for the on average 150 different physicists who use ZARAH every week
- SGI Challenge XL equipped with 18 processors for data reconstruction
- SGI Challenge DM to act as an I/O server
- AMPEX tape robot with storage capacity of 6 TB to provide our central repository for experimental and reconstructed data
- The *Open Storage Manager* (OSM) system, providing access to the STK silos where Monte-Carlo files are stored
- HIPPI links providing reliable connections with high transfer rates between the components.

This system has provided us with excellent performance, connectivity and scalability resulting in a significant improvement in data analysis at ZEUS.

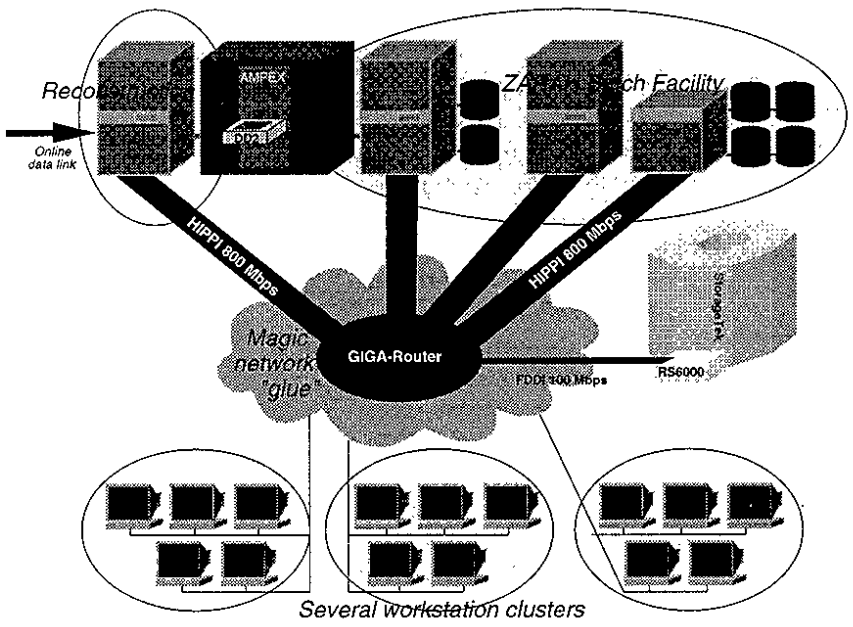

Figure 1: The *offline* computing environment in 1995

4 Interactive Desktop Environment

The ZEUS computing environment for interactive physics analysis consists of several *clusters* of personal workstations, workgroup-servers, fileservers and X-terminals connected together with the *"magic glue"* of the DESY network infrastructure. This environment provides 250 ZEUS physicists and engineers with comfortable *workplaces*. A typical *workplace* consists of:

- A personal *X Windows screen* which allows plotting histograms, using the ZEUS event display program, etc.,
- *Computing and disk storage* resources sufficient for comfortable *ntuple*-based physics analysis.
- All the *tools* necessary to develop programs, submit jobs, read mail, browse news, surf on the *Web* (we use WWW for most of the ZEUS documentation).
- A *network* which connects this *workplace* to the rest of world.

We found using *workgroup-servers* (i.e. sharing resources among a small group of physicists working on the same topic) as more efficient than using *personal* workstations. Therefore ZEUS has recently joined the AFS and UNIX-workgroup-servers projects launched by the DESY central computing group.

5 ZEUS Software

The ZEUS physics analysis software contains about 1 million lines of code (mainly FORTRAN, partially C and C++) and is being constantly developed by a large group of physicists. Code *versioning* is managed by the CERN *CMZ code management tool* and is distributed to remote sites with the *rdist* program from the central DESY repository.

In order to hide from the user differences between various platforms and give a possibility to compile, link and run the same code without changes, we have invented an *X Windows imake*-based *Makefile* generator. *Templates* used by this generator are maintained in a consistent way by our code managers.

We use *ADAMO*, an entity-relationship programming system developed at CERN to provide the underlying event structure which allows each event to be considered as a relational database.

Software Portability

In earlier times this software was developed on IBM MVS and VAX VMS platforms. In 1991 the software was ported to UNIX (first for SGI and later for DECstations). There was strong commitment in favor of inexpensive UNIX/RISC computing and ZEUS approved DECstations as the collaboration standard. Nevertheless, we realized quickly that academic prices of workstation equipment vary significantly among different countries and that rapid changes on the computer market make any long term recommendation practically impossible. Therefore we decided to maintain every release of the *offline* software for any given UNIX platform with the proviso that institutes willing to introduce a new platform should contribute to the initial port.

We found porting between various open UNIX platforms to be nearly effortless and we have to admit that a large fraction of the changes was necessary due to evident bugs rather than non-portable solutions. Today, we maintain the ZEUS *offline* software for SGI IRIX 5.3, DEC Ultrix, Digital UNIX (OSF/1), Solaris 2.x, and HP-UX. Despite the inevitable amount of work necessary to maintain all these ports we consider this approach to be truly profitable. The ZEUS collaborators have the choice of purchasing the best equipment, according to a price/performance ratio comparison, which is available on the market. ZEUS doesn't become dependent on any given hardware vendor and that leaves room for price negotiations.

6 Wide Area Networking

Most of the ZEUS experimental and Monte-Carlo data is available only at DESY. ZEUS is a collaboration of 50 scientific institutes from 12 countries and many of the ZEUS physicists may work on their analysis via the net from their home institutes. In order to provide convenient access to these data and batch computing resources at DESY the ZEUS Networking Committee was formed. They perform, on a regular basis, monitoring, and coordinate efforts in different countries, to achieve sufficient bandwidth and performance for the ZEUS networking needs. Their efforts have produced big improvements in the accessibility of DESY from outside institutes.

7 Conclusions

The ZEUS experiment has moved its physics analysis to a heterogeneous environment with an *open, scalable* and *high-performance* architecture, with a *state-of-the-art* batch facility, dedicated network-connected servers, large volume hierarchical mass storage and hundreds of interactive desktop workplaces on the net.

This environment gives us everything we need for the efficient physics analysis at ZEUS experiment. There are *no bells and whistles and flashing lights* but our main goal at ZEUS was to *do physics rather than computing* and we intend to continue our efforts with this goal in mind.

Acknowledgments

We would like to acknowledge the support of the DESY Directorate and the DESY central computing group. In particular we are indebted to Michael Ernst, Martin Gasthuber and Karsten Künne for their strong efforts on our behalf.

References

1. O. Mańczak, "The Computing Environment for Physics Analysis for ZEUS Experiment at HERA", in *Proceedings of CHEP '94 Conference*, 1994.
2. L.A.T. Bauerdick et al., "ZARAH – The Central Computing Facility for the ZEUS Experiment", in *Proceedings of CHEP '95 Conference*, 1995.
3. M. Ernst, "Putting all that (HEP-) Data to work – a REAL Implementation of an Unlimited Computing and Storage Architecture" in *Proceedings of CHEP '95 Conference*, 1995.

ZARAH – The Central Computing Facility for the ZEUS Experiment

Lothar A.T. Bauerdick, Oleg Derugin[a], Dave Gilkinson,
Matthias Kasemann, Olaf Mańczak

Deutsches Elektronen Synchrotron, Notkestrasse 85, 22603 Hamburg, Germany

In this paper we would like to present the most important technical issues concerning the design and implementation of ZARAH – the central computing facility for physics analysis for the ZEUS experiment. We would like to present to the HEP community our achievements, open problems, and ideas for the future concerning transparent access to hierarchical mass storage, indexed data access methods, data compression, ORACLE based data bookkeeping and data processing in a high performance distributed "batch" environment.

1 Introduction

During summer 1992 the HERA collider at DESY delivered its first luminosity and the ZEUS detector recorded its first data. At that time the ZEUS computing environment consisted of some few dozens of workstations (DEC and SGI), a parallel reconstruction facility running on multiprocessor SGI machines and there was a strong commitment in favor of computing solutions based on inexpensive RISC/UNIX technology. Nevertheless, a large fraction of the physics analysis and software development was done on the IBM 3090 mainframe and on the central DESY VAX cluster. Also the IBM mainframe with its STK tape library acted as our central repository for experimental and Monte-Carlo data.

Over the last three years, the architecture of the ZEUS *offline* computing evolved gradually into the *real* implementation of an *open* and scalable client-server environment with dedicated network-connected servers (storage-servers, powerful computing-servers, etc.), virtually unlimited hierarchical mass-storage and state-of-the-art network infrastructure [1]. This implementation has proven to be truly suitable for efficient physics analysis and processing huge volumes of experimental data at the ZEUS experiment. In this paper we describe the history of this evolution, our motivation and experiences.

2 History

This evolution of the ZEUS *offline* computing happened in three major steps:

- In early 1993 we implemented ZARAH *(Zentrale RechenAnlage für HERA Physik)*, our central batch facility for ZEUS physics analysis at DESY. In this first implementation ZARAH consisted of a multiprocessor SGI Power Series 460 machine ($\approx 6 \times 30$ SPECint92) equipped with some 100 GB of fast, directly connected disk storage, a direct connection to an AMPEX tape robot (256×25 GB cartridges) and the possibility to access STK silos connected to the IBM mainframe via dedicated a ULTRANET connection [2].

[a] On leave of absence from Moscow State University, Institute of Nuclear Physics

- By the end of 1993 we significantly increased the computing resources of ZARAH and the data reconstruction facility to match increasing CPU requirements by upgrading to two SGI Challenge XL SMP machines equipped with 18 R4400 processors ($\approx 18 \times 90$ SPECint92), one machine dedicated to data reconstruction and the other to batch analysis.
- In early 1995 we increased the disk storage capacity of ZARAH to 650 GB to provide fast access to the complete 1994 data sample and we increased our batch computing resources with one more SGI Challenge machine equipped with 16 processors. The disk storage upgrade included a new dedicated file-server (SGI Challenge DM with 4 processors) and all the ZARAH machines were connected with HIPPI links (800 Mbps) to a high performance switch. In addition we now have the possibility to access STK silos with the Open Storage Manager bypassing the IBM mainframe. In addition, we moved the *"online"* data link from the IBM to the reconstruction machine, effectively eliminating any dependence on the IBM.

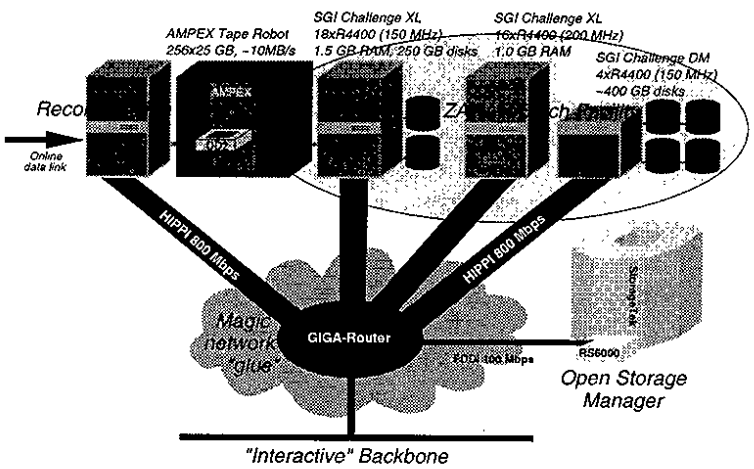

Figure 1: The ZARAH environment in 1995

To give a realistic impression about the scale and usage of this environment we would like to present the following numbers:

- Batch computing power of about 4000-5000 SPECint92 (under normal conditions we use altogether 34 R4400 processors but after reconstruction is finished we can use up to 12 additional processors) running up to 70 jobs simultaneously and processing about 700 jobs per day
- Data storage consisting of 650 GB of fast disks, 6 TBytes of directly accessible AMPEX tape storage, and fast access to STK silos. We handle above 1000 tape requests per day
- Sustained transfer rates of data from our main fileserver above 10 MB/s without visible overhead caused by accessing remote data.
- Around 150 physicists every given week using ZARAH for their analysis.

3 Motivation

In 1993 we gained some experience about the advantages and disadvantages of inexpensive workstation based solutions for offline computing vs powerful, but significantly more expensive symmetric-multiprocessor (SMP) machines.

At that time ZEUS physics analysis was done by means of an offline cluster consisting of some two dozens of DECstations 5000 ($20 \times 20 - 24$ SPECint92) and a single- backplane SGI Power Series 460 SMP machine (6×30 SPECint92). DECstations were equipped with some 2-3 GB of disk space each interconnected via Network File System (NFS) while the SGI machine had some 100 GB of fast direct disk storage and a dedicated ULTRANET connection to the IBM-STK tape library where the data was kept.

We found inexpensive distributed workstation solutions very useful for CPU-bound computing (e.g. editing, histogramming, mail), but found workstation based data analysis to be painful and slow. Major bottlenecks were:

- The necessity of shipping large datasets from the remote data repository to workstation disks over the network; with average file sizes of 200 MB, an effective transfer rate of 100 KB/s (an Ethernet backbone was shared by 20 workstations) and a limited number of drives in the STK silos, users could often wait over an hour for a dataset to arrive.
- Datasets stored temporarily on a distributed network filesystem were accessible at similar transfer rates so that a typical analysis job, which running on a 20-25 SPECint92 workstation able to process 7-8 events/s (35 KB per event), was blocked by insufficient network I/O rates.

Meanwhile, analysis done on the SGI machine avoided these bottlenecks and gave the possibility to perform a pass through the whole disk data sample within one night. Given these observations, we decided to separate the functions of workstations and SMP machines. The ZEUS workstations would in future be used for interactive analysis involving editing, histogramming, *background* Monte-Carlo production [3] etc., and the SMP machines would be used strictly as batch machines providing fast access to data and efficient use of resources. Below we discuss some of the methods we have developed to reach our present configuration.

4 The ZARAH Batch Environment

For historical reasons, we use the *Network Queuing System* (NQS) as a batch control system. Additionally, we have implemented a job spooling system and a set of simple client- server tools, whereby the user can submit a job to the batch system directly from his workstation, monitor its execution and retrieve results from the batch system to his workstation.

This approach has proven to be very efficient and allows us to minimize interactive usage of the batch platform. Interactive use is not restricted but *"not recommended"*. We have learned that it is highly reasonable to give users some freedom concerning usage of the batch system and monitor that this freedom is not misused rather than preemptively restrict *user rights*.

In addition, as part of the batch package, we have invented smart SIGSEGV and SIGBUS *signal handler* routines which can spawn the *dbx* debugger, execute a debugger command file which prints a useful debug report, and finally calls a user exit routine in order to save results. These signal handlers eliminate the need for running long jobs through the debugger and provide useful diagnostics for the user.

5 Data Handling

The problem of data duplication is a well known issue in HEP data handling. Essentially, there are always two *wishes* causing this problem:

- select smaller subsample of tape-data to keep it on *faster* media (e.g. disk) to reduce access time,
- select subsample of *"interesting"* events from a large data volume to avoid sequential processing of all the data.

We aimed to organize the hierarchy of the storage in a proper way, to use fast and randomly accessible *disks* efficiently and to eliminate data duplication. Therefore,

- we created MINI-DST datasets with *"smaller"* events (35 KB vs. 100-150 KB in reconstructed datasets) containing only the most important (but sufficient for most analysis) information to store more data on disks,
- we compressed MINI-DST datasets (on a *per-event* basis) to reduce the amount of disk storage required for *one year's data sample* to some 400-500 GB,
- and we invented the *ZEUS event directories* i.e. an indexed data access method which provided us with random access to any given event.

6 Data compression

We have investigated various possibilities of efficient data compression. We found that predictive algorithms based on the knowledge of event structure are difficult in implementation and they have to be changed if the event structure changes. Results obtained with the Lempel-Ziv (LZ77) non-predictive dictionary compression algorithm [LZ] (we have used implementation derived from the GNU gzip program) were much better. We achieved an average compression factor (i.e. ratio of compressed size to uncompressed size) of 65% with relatively small penalty in the data processing rate.

7 Tape staging

At ZEUS, large volumes of data are created only by a few privileged users during data reconstruction and reprocessing and most of the users use this data as read-only. The frequency and method of accessing given datasets depends on the type of data they contain. The ZEUS *data* can be classified as:

- raw data – about 2 TBytes a year
- reconstructed data (RDST) – about 2 TBytes a year
- MINI-DST – about 0.5 TBytes a year (compressed)
- Monte-Carlo – sample of several TBytes

The raw data is accessed during reconstruction and reprocessing and then access is rather rare.. The RDST datasets are accessed only if analysis requires information which has been stripped and can't be found on MINI-DSTs. The RDST datasets are usually accessed only within the first 1.5 years. Access to MINI-DST datasets can be easily optimized by a permanent disk storage dedicated to the complete last year's sample and small subsamples of selected events (runs) from previous years. We found as well that it is virtually impossible to create this kind of disk *cache* for Monte-Carlo datasets since the amount of randomly accessed data has a size of several TBytes.

In order to make high-volume tape-based data analysis efficient it is essential to guarantee sufficient I/O bandwidth and reasonably short access time. These requirements can be easily translated into the number of simultaneous staging streams *(drives)* and transfer rates per stream. Our experience with accessing IBM-STK silos via ULTRANET (14 streams but low transfer rates) and AMPEX tape robot (transfer rate up to 10 MB/s but only one stream) showed that both these requirements are very important. The OSM system has greatly improved reliability and performance of our tape staging system.

We believe as well that it is crucial for efficient *tape*-based data processing to make the staging process *fault-tolerant* and *transparent* to the user. For this purpose, we are about to implement an additional layer — a *virtual meta-data filesystem*, visible to the user through an NFS-compatible interface, and an *open server*, performing the actual file staging and returning to the user process an opened file handle. The user can access the *virtual meta-data filesystem* via standard Sun Microsystems NFS protocol but returned information is generated by the *meta-data database* server and points either to an existing dataset available in a temporary staging pool or to an open server named socket. The *open server* retrieves information from the *meta-data database* about the size and physical location of the dataset, allocates the required disk space, transfers from tape to disk and performs all the necessary error handling. By this method staging is hidden from the user process and looks very much like access to regular disk files (an emulation of the open system call, aware of the open server, is implemented in an additional dynamically linked run-time library). A prototype of this system has been already implemented and we expect to introduce it into full operation soon.

8 Documentation and Databases

Having fast, powerful and efficient computing resources is certainly a necessary condition for physics analysis at ZEUS. However, all this is wasted if no one can use it.

We have found it extremely useful to have good documentation available for both the new and experienced users. New users are provided with working examples of programs and instructions about what to do when their results don't match those documented. Experienced users are kept up to date on new developments and changes and have access to a *FAQ* about the system, software and data. Our initial hardcopy manual has been superceded by a HTML version which allows us to do *"real time"* updating, something that is often neglected with text documents.

Information about data is also an important topic. The ZEUS data contain several millions experimental and Monte-Carlo events and consists already of some 70,000 datasets. In order to use and maintain in a consistent way information about relations between runs and datasets, about parameters use for generation of Monte-Carlo datasets etc, we have designed and implemented, using the *ORACLE* RDBMS, some relational databases. The database information is updated automatically during data reconstruction and production of new Monte-Carlo datasets and is available to all the ZEUS physicists with user friendly client-server tools. Our initial experience with *ORACLE* has been quite good and we intend to continue our database development in this direction.

9 Future plans

Aside from ongoing efforts mentioned in the previous sections there is a need (and some ideas how to do it) to implement transparent job *checkpointing*. We believe very much that an ability to suspend any given batch job at any time and later resume the process, possibly on a different machine, is very desirable. *Checkpointing* is one of the most important issues concerning reliable and distributed batch processing.

In addition, we plan to increase our disk space capacity by approximately 100 GB a year in order to keep selected data taken in the former years.

10 Conclusions

The joint efforts of ZEUS and the DESY central computing group [4] have built a state-of-the-art facility for batch analysis. Our recipe consists of simple design, an open technology with scalable architecture, some *thinking* about efficient ways of data processing and some effort to put it all together. ZARAH, with virtually unlimited hierarchical mass-storage, powerful computing resources and easy access from the interactive desktop computing facilities on the net, has become a useful and important tool for the efficient physics analysis at the ZEUS experiment.

We believe that our motivation and experience in developing this highly successful system can be useful for experiments of the year 2000 and beyond.

References

1. L.A.T. Bauerdick et al., "The Physics Analysis Environment of the ZEUS Experiment", in *Proceedings of CHEP '95 Conference*, 1995.
2. O. Mańczak, "The Computing Environment for Physics Analysis for ZEUS Experiment at HERA", in *Proceedings of CHEP '94 Conference*, 1994.
3. B. Burow, "Funnel: Towards Comfortable Event Processing", in *Proceedings of CHEP '95 Conference*, 1995.
4. M. Ernst, "Putting all that (HEP-) Data to work..." in *Proceedings of CHEP '95 Conference*, 1995.

STATUS OF THE ZEUS EXPERT SYSTEM (ZEX)

Ulf Behrens, Mariusz Flasiński, Lars Hagge, Janusz Jurek, Kars Ohrenberg

Deutsches Elektronen-Synchrotron, Notkestrasse 85, 22603 Hamburg, Germany

The expert system ZEX is supporting the shift crew in operating the ZEUS detector. It helps to increase both efficiency and reliability of experiment operation while reducing the required experience and expertise of the shift crew. The task of the expert system is to detect anomalous behavior in any part of the experiment, to trace errors to their origin, and to recover or help to recover the system as quick as possible. ZEX is based on the commercial expert system shell RTworks. The large number of human experts in the ZEUS environment requires the knowledge to be stored in an intuitive way, and a rule based system was considered to be the only appropriate way of knowledge storing. ZEX is put on top of existing systems, and the layered structure of ZEX reflects the structure of the online system. The Slow-Control sub-expert system is running since 1994, several parts of the DAQ and the DQM sub-expert systems have been implemented since then.

1 Introduction

The ZEUS detector is controlled and read out by a highly parallel distributed online system [1]. To increase both efficiency and reliability of experiment operation and at the same time reduce the required efforts and expertness of the shift crew, an expert system project was launched by the ZEUS collaboration in July 1992 [2]. The goal was to automatically detect anomalous behavior in any part of the experiment, to trace errors to their origin, and to help to recover the system as quick as possible.

In a first step, an expert system prototype (ZEX-P) was created which processed monitoring information from a subsystem of the data acquisition system, the Eventbuilder [3,4]. The prototype was designed using SA/SD-methods, and was implemented in the programming language C, based on syntactic pattern-recognition algorithms. Experience regarding feasibility and performance was encouraging, however it became clear that keeping a full-blown expert system maintainable would require for dedicated programming languages and development tools [5]. ZEX-P was therefore re-engineered in a rule-based approach, before it was finally decided to implement ZEX based on the commercial expert-system shell RTworks [6,7].

2 Overview of ZEX

General Overview

Fig. 1 shows the architecture of ZEX. The structure of ZEX was chosen to reflect the structure of the online system. Three dedicated sub-expert systems evaluate and thus encapsulate knowledge of distinct domains, while a top-level module analyses the output of the sub-expert systems to derive the overall system characteristics.

The different modules communicate through a global hierarchical data structure called blackboard. The blackboard is hierarchically organized into information levels corresponding to levels of data processing, which range from raw monitoring information to derived system status information. The problem-solving knowledge

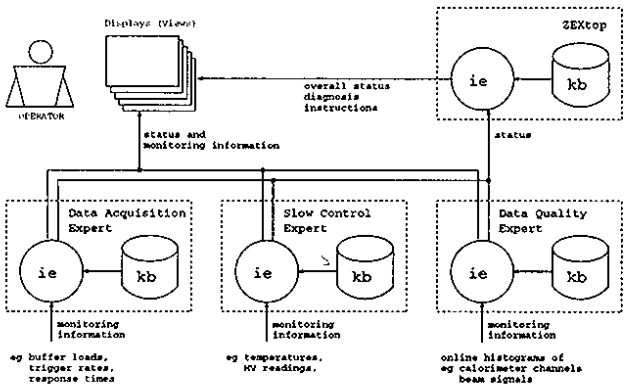

Figure 1: Architecture of the ZEUS Expert System (ZEX).

is modularized and encapsulated in so-called Knowledge Sources, thus partitioning the Expert System according to the structure of the experiment. The majority of the Knowledge Sources is implemented as rules, however the Blackboard architecture has enabled us to include some of the pattern recognizers which have been developed for ZEX-P [8]

Sub-Expert Systems

The slow control sub-expert-system monitors the basic hardware of the experiment. It surveys the states of power supplies, racks, crates, photomultipliers, temperature and radiation sensors, cooling, etc. Based on this information, it decides if efficient data taking is possible, and proposes what to do to regain an acceptable state.

The data quality sub-expert-system tries to ensure high quality of the data written to tape. It monitors the beam quality and background rates, and it checks the data from the major components for miscalibration, dead channels, etc. The data flow sub-expert-system surveys the data taking. It monitors the central components of the data acquisition system (ie front-end systems, trigger stages, data storage task) for data rates, deadtime, response times etc. At the top level, ZEX is combining the information about the subsystems to an overall system understanding. It selects the most important information and presents it to the shift crew.

Status of ZEX and Experience

The Slow-Control Expert was the first sub-expert system to be put into operation during the 1994 data taking period. Its functionality has been expanded steadily. Finally, first parts of the other subsystems have been added.

For the 1995 data taking period, ZEX is tuned to speed up the procedure of expanding and modifying the knowledge of the system, thus enabling fast reactions to changes in the run conditions, and preparing ZEX to handle short-term conditions in the ZEUS environment.

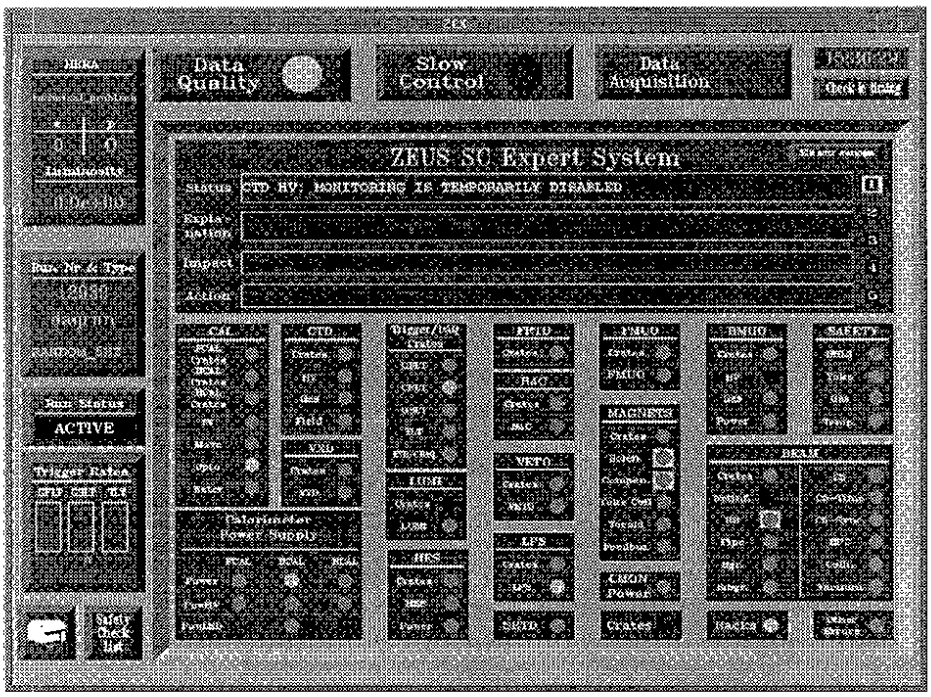

Figure 2: The ZEX Slow-Control display.

Development of ZEX started only after the experiment was put into operation, hence ZEX had to be put on top of existing systems. The major constraint for the development of ZEX was not to interfere with the operation of the online system, which introduces difficulties since very often information which is essential for an accurate reasoning has shown to not be accessible to ZEX.

3 Impact on future experiments

In a fast-changing environment like a HEP experiment, where experts are not always available at the location of the experiment and are frequently changing their positions, an expert system provides a central knowledge repository, which makes information available to anybody whenever (maybe also wherever) it is needed. Parts of the experiment operation (eg data acquisition operation, readout configuration etc) can even be automated with an ES.

A mandatory condition for successful operation is the availability of monitoring information from and access to the different detector parts, their readout and the trigger system. It is thus beneficial to incorporate an expert system already at very early stages into the experiment design (protocols, network layout etc).

Knowledge engineering and artificial intelligence are branches of computer science which are mostly alien to high-energy physicists. It should thus be considered to involve computer scientists into the design and development of expert systems

in HEP.

References

1. C. Youngman, "The ZEUS Data Acquisition System", in **Proc. CHEP'92**(CERN 92-07)
2. U. Behrens, M. Flasiński, L. Hagge, "ZEXP - Expert System for ZEUS, Problem Analysis, Theoretical Background and First Results", (DESY 92-141)
3. U. Behrens, M. Flasiński, L. Hagge, W. O. Vogel, "The Eventbuilder of the ZEUS Experiment", in **Proc. CHEP'92**(CERN 92-07)
4. U. Behrens, L. Hagge, W. O. Vogel, "The Eventbuilder of the ZEUS Experiment" *Nucl. Instrum. Methods* A **332**, 253-262 (1993)
5. U. Behrens, M. Flasiński, L. Hagge, K. Ohrenberg, "ZEX - An Expert System for ZEUS", contributed to *8th Real-Time Comp. Appl. in Nucl., Part. and Plasma Phys. (RT93)*, Vancouver 1993, IEEE Trans. Nucl. Sci. **41**, 152-156 (1994)
6. U. Behrens, M. Flasiński, L. Hagge, K. Ohrenberg, "Paradigms and Building Tools for Real-Time Expert Systems", in **Proc. CHEP'94**(LBL 35822), San Francisco 1994
7. *RTworks V3.0 (User Documentation)*, Talarian Corporation, Mountain View, CA., 1994
8. U. Behrens, M. Flasiński, L. Hagge, J. Jurek, K. Ohrenberg, "Recent Developments of the ZEUS Expert System ZEX", inv. talk at *9th Real-Time Comp. Appl. in Nucl., Part. and Plasma Phys. (RT95)*, Lensing (MSU) 1995, to be published

FUNNEL: TOWARDS COMFORTABLE EVENT PROCESSING

Burkhard D. Burow[a,b]

Universität Hamburg, II. Institut für Experimentalphysik,
Luruper Chaussee 149, 22761 Hamburg, Germany

The funnel software package has solved for the ZEUS collaboration the problem of Monte Carlo event production; a problem faced by many HEP experiments. Thanks to extensive automation, a few man-hours per day are sufficient to resolve problems and to manage the entire ZEUS Monte Carlo production. Other than specifying the events to be produced, ZEUS physicists are thus freed from the chore of Monte Carlo production. As an additional benefit, the computing cycles required for production are nearly cost free since they replace otherwise idle cycles on hundreds of unix workstation and server computers, with minimal interference for their regular users. The computers are spread across a dozen sites around the world and continually deliver the effective equivalent of approximately one hundred dedicated computers.

Funnel successfully demonstrates that generic independent tools can provide comfortable event processing. With an emphasis on automation and fault-tolerance, the tools manage all aspects of event processing including the job queues, the execution and failures of the processing program, parallel processing, as well as data buffering, archiving and remote transfer. The L3, HERMES and H1 collaborations are presently creating Monte Carlo production systems, using the funnel experience and, to different extents, parts of the funnel software package.

The experience gained with funnel encourages the construction of EVPRO, a general purpose software package for event processing. EVPRO would build on top of existing software; for example CPS or PVM for parallel processing. Whether on a dedicated farm of computers or using idle cycles, an application of any size could then easily enjoy the comfort of automated, fault-tolerant event processing. EVPRO aims to minimize application-specific event processing software, whose high development costs can only be justified for the largest of applications. A casual user may provide EVPRO with only the processing program and the data to be processed. A more complex or real-time application would tune EVPRO to its needs; for example, integrating custom hardware for the flow of event data. Making optimal use of the available computing resources, EVPRO would manage all aspects of the event processing. Monte Carlo production, event reconstruction and software triggers could use EVPRO, as could any computing application, inside or outside of HEP, which can be expressed in terms of events.

In principle, event processing is a solved or even a trivial problem. Given an implementor, EVPRO could provide the trivial solution in practice.

1 Introduction

The ZEUS experiment at DESY records of order 10^7 events per year at the HERA electron-proton collider. The physics analyses require several times more Monte Carlo (MC) events than HERA events; on average $5{\times}10^7$ MC events/year $\approx 10^6$/week $\approx$ 100/minute. Due to the large number of analyses, the required MC events are distributed across approximately 100 job requests per week. The MC simulation of an event typically consumes one minute of workstation CPU. Thus, ZEUS MC production effectively requires

a. Presented in the plenary session 'F. Systems and Facilities' at Computing in High Energy Physics (CHEP95), Rio de Janeiro, Brazil, 18-22 September 1995.

b. Also supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) and by the BMBF under Contract Nr.056HH29I.

continual use of 100 dedicated workstations. With an average MC event size of 60 Kbytes, the total production rate of 100 Kbyte/second can be carried by local area networks (LAN), though not generally on wide area networks (WAN). An aggregate production of about 60 Gbytes/week $\approx$ 3 Tbytes/year must be archived.

1.1 Resources for ZEUS MC Production

There are approximately 500 physicists in the ZEUS collaboration, providing access to more than 500 workstations spread across sites in Europe, North America and Asia. Their otherwise idle CPU time is harnessed to effectively provide the 100 dedicated workstations required for MC production. At each site, the LAN connecting the 5 to 90 workstations allows MC production to use parallel processing. All the sites may be remotely controlled using ftp/telnet/rsh. The WAN, idle night bandwidth permitting, or Exabyte tapes sent by courier mail[1] transport the produced events to the MC archive at DESY.

The above resources are managed by funnel to provide the ZEUS MC facility.

1.2 The ZEUS Physicist's View of Funnel

Physicists submit jobs to funnel. Each job provides the input events, in terms of particle momentum vectors, to be sent through the MC simulation specified by the job. The requested detector and trigger simulation and event reconstruction may range from one of the standard ZEUS configurations through to a physicist's private development software.

Physicists receive from funnel for each job the archived output events, log files and information on any input events which crashed the simulation software.

ZEUS physicists are thus freed by funnel from the chore of MC production.[2]

2 Global ZEUS MC Production

2.1 Transferring Gbytes Around the World

Each input job is sent over the network to one of the production sites for processing. To transfer data to a remote site, it is placed into a local *transfer directory*. This is an example of a *daemon directory*. It is monitored by a *daemon process*, which runs in the background and acts on every file appearing in the directory; logging the files, the actions and the exceptions handled. A successful action is noted by deleting the file or moving it elsewhere. Faults beyond the daemon's exception handling are e-mailed to the user. The action performed by the daemon is thus asynchronous, modular, automated and fault-tolerant. Once a process places a file in a daemon directory, the file requires no further actions nor exception handling from the process; the daemon effectively guarantees success.

Depending on the site, the output data is transferred to the MC archive by the **farc** daemon, using Exabyte tapes, or by the **fmvbat** daemon, using ftp.

2.2 The Job Queue at Each Site

To have an input job processed, it is placed into the local *job queue directory*. The job queue daemon ensures that for each job, the output data appears in the archive or, if the job fails in a manner beyond the daemon's exception handling abilities, e-mail notifies the funnel operator. For example, the daemon restarts a job aborted by a computer reboot and notifies the funnel operator if an input job is corrupt.

3 Comfortably Processing a Single Job

In an ideal world, with infinite computer speed, unlimited disk space and bug-free programs, processing a single job is trivial. The processing program is simply run on the input events in order to produce the output events.

In the real world, event processing is not trivial. A job may take days to complete and produce many Gbytes of output, but neither time nor space are generally available. In addition, others may want to use the computer, the computer or disk or network may crash or for some of the input events, the simulation program may crash or enter an infinite loop.

The funnel tools can't provide the ideal world but, as the following outlines of some of the major tools show, they can provide its comfort to good approximation.

3.1 Buffering Gbytes of Output

The output data produced by a job is buffered on disk during processing. The buffering is performed automatically by the following two funnel tools. Once the job has completed, symbolic links effectively place the buffered data into a transfer directory of section 2.1.

The **getBUFFER** utility manages free disk space. In response to a process' request, getBUFFER returns an address on a disk with free space greater than the current request plus outstanding allocations on that disk. Thus, the allocated space is guaranteed to be free for use by the process. getBUFFER automatically reclaims allocations outstanding to processes which have exited. getBUFFER does not delete 'old' data; that's left up to the application. Thus one or more processes may easily use multiple disks.

The **dogB** utility places a process' output into a getBUFFER area or areas. dogB uses Unix I/O to block the process until the getBUFFER area is available. Thus, the process is gracefully blocked while the disks are full, resuming once there is free space.

Thanks to getBUFFER and dogB, the process effectively enjoys unlimited disk space.

3.2 Managed Event Processing and Parallel Event Processing

The real world problems of job processing, those not related to disk space, are solved by making use of the *natural unit of processing*, the *event*. The input data consists of independent events, each of which is independently processed to produce an output event.

The solution keeps the simple original processing program, schematically shown in Figure 1a), but introduces the manager program as shown Figure 1b). The event is the manager's unit of input and output (I/O). By manipulating entire individual events, the manager is generic across different processing programs and can solve the problems of event processing. For example, if using the idle cycles of a computer, the manager recognizes when other users become active and then terminates or suspends the processing program; resuming event processing once idle cycles are again available.

Bugs are inevitable in large computer codes such as the ZEUS simulation programs. While bugs are eventually fixed, they are often irrelevant and should thus not halt event processing. In analogy, HEP experiments don't usually halt data taking just because an online program occasionally crashes. Therefore, if the processing program crashes, the current event and the stack trace are recorded by the manager, which then continues processing. As described in section 1.2, a job's output data is accompanied by the crash information. The physicist can thus determine the relevance, if any, of a crash and can pursue the code's author(s). Similarly, if the processing program enters an infinite loop, the

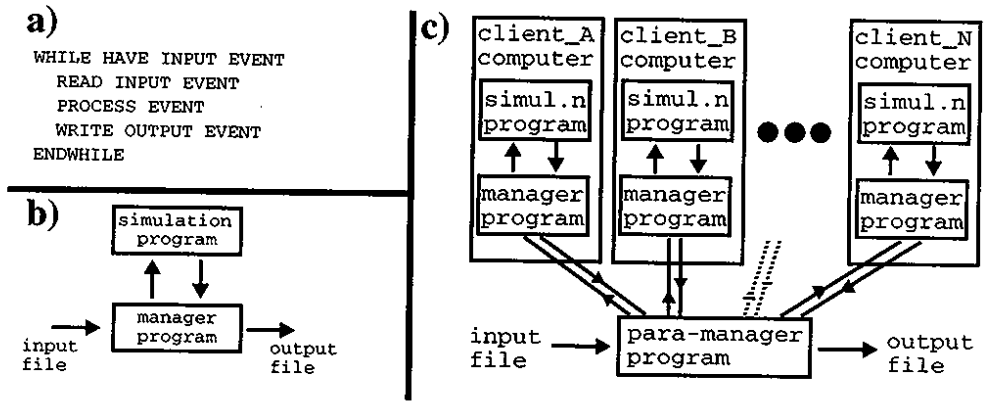

Figure 1 a) Pseudocode description of an event processing program.
b) Event exchange with manager replaces direct access to input and output files.
c) Parallel event processing using the generic manager and para-manager tools.

manager e-mails the author(s), who can then examine the running loop with a debugger. The processing program is eventually forced to crash and is treated as described above.

The independent events may obviously be distributed[1] across many copies of the processing program. As illustrated in Figure 1c), the para-manager program easily implements this trivial parallelism. As for the manager program, by manipulating entire events, the para-manager is generic across any processing program. Intrinsically simple, the para-manager provides various features such as the dynamic connection and disconnection of clients, allowing the use of idle cycles for processing.

4 EVPRO for Comfortable Event Processing

For event processing, be it MC production, a software trigger or event reconstruction, HEP is concerned with the input and output events, the processing software and any exceptions. HEP has no intrinsic interest in managing the computer resources passing the events through the processing software. As outlined in sections 2 and 3, the funnel architecture demonstrates that generic, independent tools can manage the computing resources required for event processing. Going beyond ZEUS, funnel thus encourages the construction of EVPRO, a general purpose software package for event processing. EVPRO could benefit HEP tremendously, since event processing is currently managed manually and/or using application-specific software. This conference alone provides many examples[3], in addition to funnel, of the HEP energy currently spent managing event processing; energy to be spared by EVPRO. Interest in EVPRO is corroborated by the L3, H1 and HERMES interest in funnel mentioned in the abstract. At present, EVPRO does not exist and seeks an implementor(s).

EVPRO would consist of event processing tools, containing some new software, and information helping the user determine the solution to the event processing problem. The solution may involve user, EVPRO and other software. The other software, which EVPRO would also heavily use, could include for example CPS or PVM for parallel processing, CERN's SHIFT utility for disk space management and NQS or LSF for job management.

Passing a file of events through a simulation program demonstrates a simple use of EVPRO. After relinking the simulation program to use the EVPRO manager's event reading and writing routines and providing a routine identifying the input data event boundaries, the physicist would be freed of the details of the event processing, including parallel processing, disk management and most exception handling. For many such jobs per day, EVPRO would provide the tools required for a funnel-like MC production facility.

HEP software triggers using distributed event processing could be managed by EVPRO as illustrated in Figure 2a). The hardware for event I/O is integrated into the manager program which shares event memory with the trigger program.

Outside of HEP, many applications are similarly divisible into events or tasks, for which, the calculation time is greater than the data transfer time. As shown in Figure 2b), EVPRO could provide comfortable master-worker parallelism to these applications.

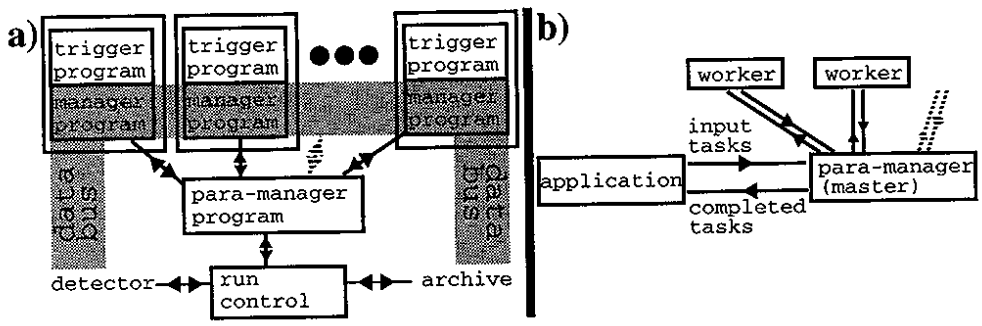

Figure 2 a) EVPRO for a HEP software trigger. b) EVPRO for generic master-worker parallelism.

Summary

Freed by funnel from the chore of MC event processing, ZEUS physicists have more time for physics. In addition, funnel provides inexpensive MC production using idle cycles.

Implemented as EVPRO, funnel's idea of generic independent event processing tools should be portable to HEP and other applications. Freed by EVPRO from the chore of event processing, HEP should then have more energy for physics.

Acknowledgments

Marc St-Laurent created the farc tape daemon. Wouter Verkerke created the software managing the global flow of jobs. ZEUS MC production and the funnel software owe much to F. Benard, L.W. Hung, L. Wai, R. Yoshida, J. Butterworth, M. Kuze, D. Gilkinson, A. Bruni, M. Botje and H. Uijterwaal. Strong support comes from the DESY computing services and from the many ZEUS colleagues at the production sites and in the ZEUS software groups. Special thanks to ZEUS for all its otherwise idle computing cycles.

References

1. The HEP precursors of techniques such as data transport by tape and event farming are described in the proceedings of previous CHEP conferences.
2. S.W. O'Neale, CHEP92, p.471, describes a similar service for OPAL.
3. M. Meesters. E. Anderson. A. Gellrich, R. Itoh. J.Silva. F. Gagliardi. All at CHEP95.

MIXED LANGUAGE PROGRAMMING

Burkhard D. Burow[a,b]

Universität Hamburg, II. Institut für Experimentalphysik,
Luruper Chaussee 149, 22761 Hamburg, Germany

Computing in the next millennium will be using software from this millennium. Programming languages evolve and new ones continue to be created. The use of legacy code demonstrates why some present and future applications may span programming languages. Even a completely new application may mix programming languages, if it allows its components to be more conveniently expressed. Given the need, mixed language programming should be easy and robust. By resolving a variety of difficulties, the well established cfortran.h package provides the desired convenient interface across the C and Fortran programming languages.

This presentation examines mixed language programming. It aims to help programmers of all languages benefit from the possibilities offered by mixed language programming and to help them create software which in turn may enjoy a long and useful life, perhaps at times in a mixed language program. By encouraging and facilitating code reuse and the use of well-suited programming languages, one may help eliminate the qualifier in the maxim:

"Scientists stand on the shoulders of their predecessors,
except for computer scientists, who stand on the toes."

1 Introduction

1.1 Applications from Components

A computer application consists of components, each supplying part of the action performed by the application. The action provided by any given component may be required by several applications. Therefore, applications reuse components. The expensive alternative to reuse is to recreate components.

1.2 The Need for Mixed Language Programming

A component is coded in a programming language. An existing component is thus coded in one of the various programming languages. Depending on the action performed by the component, it may be best expressed in a particular language. For a component coded in one of the general purpose programming languages, the choice of programming language is strongly determined by the programmer's preferences. The components used by an application may thus span several languages. The creation of such a mixed language application is called Mixed Language Programming (MLP).

1.3 Mixed Language Programming is NOT Translation

For some components and/or languages, it may be possible to translate the component into another language. Translation is sensible when further development and maintenance of the component will take place in the new language and is abandoned in the old language. Such a translation is not an MLP issue.

a. Presented in the parallel session 'E.1 Languages' at Computing in High Energy Physics (CHEP95), Rio de Janeiro, Brazil, 18-22 September 1995.

b. Also supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) and by the BMBF under Contract Nr.056HH29I.

Translation is generally not a method to perform MLP, since MLP is concerned with the use of a component and thus with only its interface or 'surface'. There is no need to translate the 'body' of the component. Additional practical difficulties include:

- The source code of the component may not be available. For example, many commercial components are only available as libraries of compiled objects.
- Due to the different abilities of the languages involved, the translation may be practically impossible. For example, the translation of a component written in C into Fortran is practically impossible, since C has many features not present in Fortran.
- Even where possible, translation is difficult, even with tools.
- Translation introduces opportunities for bugs.

1.4 The Anatomy of a Component

The anatomy of a typical component is sketched in Figure 1a). Within a programming language, the use of components is well-supported: the name is obvious; passing input and output data is easy and there is a consistent environment for any component side-effects.

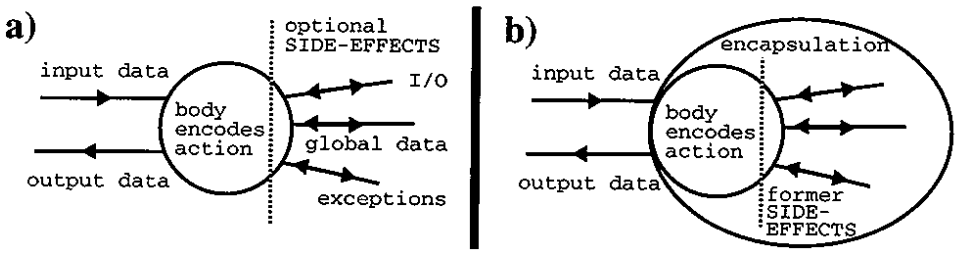

Figure 1 a) The anatomy of a typical component. b) Effectively eliminating side-effects by encapsulation.

1.5 The Trouble with Mixed Language Programming

Crossing languages, the use of components is generally NOT well-supported: a component's name may not be obvious; passing input and output data may be complicated and there is a potentially troubled environment for any component side-effects.

1.6 Avoiding Trouble with Side-Effects

Following good programming practice, side-effects are best avoided since they are generally trouble-some, not just for MLP. For example, instead of printing an error message, a component should return an error code among the output data. As sketched in Figure 1b), encapsulation may allow an existing component to be effectively freed of side-effects.

2 Three Types of Mixed Language Programming

2.1 Coarse-Grained Mixed Language Programming

For coarse-grained MLP, the application consists of multiple executables. Each executable consists of components written in a single language. The basic notion is that each executable works on an intermediate file of data. Once the work has completed, the next executable may work on the file of data. The scheme can enjoy two refinements. Scripts can automate running the executables. The intermediate file(s) can often be replaced by a flow of data through a pipe or pipes connecting the executables.

Coarse-grained MLP has several advantages and is an obvious method if one of the executables already exists. It has none of the MLP problems, other than passing data between executables written in different languages. Not just for MLP, but in general, the separate executables should simplify the development and maintenance of the application.

Coarse-grained MLP is unfortunately of limited applicability. It is unsuitable for applications which thickly interleave the use of components across different languages.

2.2 CORBA

The Common Object Request Broker Architecture (CORBA) is a powerful, generic environment for combining components. For component use, the language of the component is irrelevant once the language is bound to the Interface Definition Language (IDL). The IDL solves the MLP problems for component names and for passing input and output data. Experience with CORBA for MLP is presented at CHEP95 by Quarrie.

2.3 Mixed Language Executables

A Mixed Languages Executable (MLX) is perhaps the most common form of MLP. For example, C and Fortran routines are combined to create CERN's popular PAW[1] program.

Unfortunately few programming languages provide standard support for MLX. An exception is C++, which supports the use of C routines. In future, Fortran 2000 may provide standard 'interoperability with C'.

Despite the dearth of standard support, MLX programming is possible. The remainder of this presentation examines the difficulties of MLX and their solution.

3 Mixed Language Executables

3.1 Direct MLX ?

In the most direct form of MLX, the machine dependent recipe to call a foreign routine is contained in the application code for each foreign routine and for each machine. As demonstrated by the code fragment in Figure 2a), direct MLX can be tedious, cluttered and error-prone. Moreover, the troubles geometrically worsen with the complexity of the routines' interfaces and with the number of machines and calls to foreign routines. Nevertheless, direct MLX may be tolerable for applications running on a single type of machine, using routines with simple interfaces or when nicer methods, see below, are unavailable.

3.2 Wrappers for MLX

As illustrated in Figure 2b), all MLX problems disappear for the application programmer once a wrapper provides a native interface to the foreign routine. A native interface implies that the use of the foreign routine via the wrapper is machine independent, with a simple name and easy passing of input and output arguments. For any foreign routine, a wrapper has only to be created once for each machine type.

3.3 Direct Wrappers ?

Wrappers may of course be directly created by the programmer, but this shares many of the troubles of direct MLX. As illustrated in Figure 2c), considerable tedious effort has to be exerted to create a wrapper for each routine and machine.

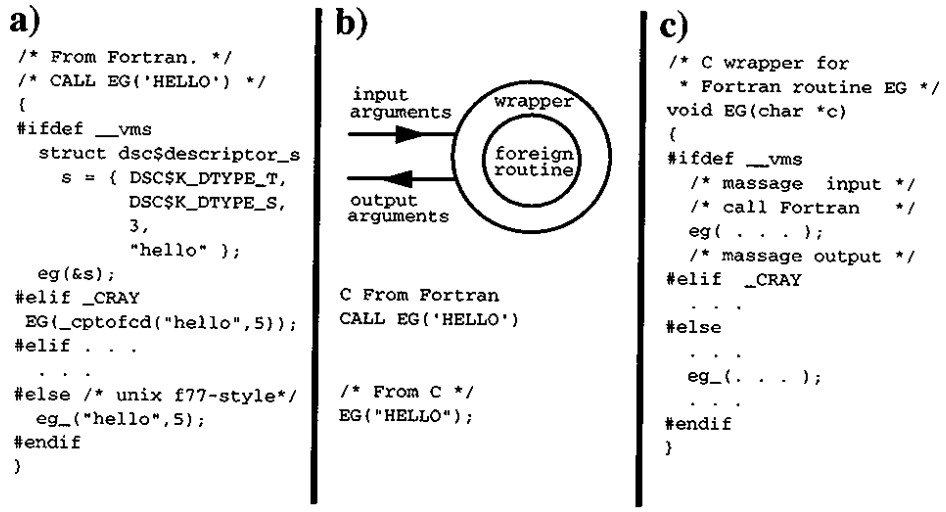

Figure 2 a) An example code fragment of C calling Fortran using direct MLX.
b) An illustration of using a wrapper for MLX.
c) An example of a direct wrapper to allow C to easily call a Fortran routine.

3.4 Machine Independent Wrapper Specification

A wrapper may be easily created once a machine's MLX recipe has been encoded in some form. As illustrated in Figure 3a), the creation of the wrapper thus only requires the encoding and a machine independent specification of the wrapper.

3.5 cfortran.h

MLX recipes have been encoded into various 'wrapper schemes'. For combining C and Fortran[2], cfortran.h[3] is the most complete scheme. It is available via anonymous ftp at zebra.desy.de. cfortran.h supports:

- C calling Fortran. Fortran calling C.
- Names of routines and common blocks.
- Input and output arguments. Function return values.
- Most data types and is extensible with simple user defined types.
- VMS, OSF, Ultrix, AIX, IRIX, CRAY, SunOS, Apollo, HP-UX, Convex, . . .

3.6 hbook.h

The wrapper specifications using cfortran.h are typically in a C header file. A popular example is hbook.h, fragments of which are shown in Figure 3b). As demonstrated by the example C program in Figure 3c), hbook.h allows C to easily access CERN's HBOOK library of Fortran routines. C access to other CERNLIB Fortran libraries is provided by adamo.h, geant321.h, minuit.h, etc. The f2h utility in CERNLIB can 'automatically' create the *.h header files from Fortran source code.

a)

```
#include "cfortran.h"     /* encoding */

      /* machine independent wrapper */
#define EG(C) \
  CCALLSFSUB1(EG,eg, STRING, C)

main()
{
EG("HELLO"); /* call Fortran routine */
}
```

b)

```
/* hbook.h of cernlib@cern.ch */
. . .
#define HBARX(A1) \
        CCALLSFSUB1(HBARX,hbarx,INT,A1)
. . .
#define HISTDO() \
        CCALLSFSUB0(HISTDO,histdo)
. . .
```

c)

```
/* An example of HBOOK use from C. */
#include "cfortran.h"
#include "hbook.h"

#define NWPAWC 50000
typedef float PAWC_DEF[NWPAWC];
#define PAWC COMMON_BLOCK(PAWC,pawc)
COMMON_BLOCK_DEF(PAWC_DEF,PAWC);
PAWC_DEF PAWC;

main() {
  int i, id=10;

  HLIMIT(NWPAWC);
  HBOOK1(id,"1-DIM",100,1,101,0);

  for ( i=1; i<=100; i++)
    HFILL(id, i+.5, 0, 10*(i%25) );

  HISTDO();
}
```

Figure 3 a) An example C program creating and using a machine independent wrapper specification.
b) Fragments from the C header file hbook.h, providing C wrappers to the Fortran HBOOK routines.
c) An example C program using CERN's HBOOK library of Fortran routines.

3.7 HERMES and cfortran.h

The HERMES collaboration at HERA is bilingual, though most programmers write either C or Fortran. As of one of the largest and most successful HEP users of cfortran.h, HERMES enjoys comfortable and productive use of both C and Fortran throughout their software development including the use of outside libraries and the creation of HERMES libraries and applications[4].

4 Conclusion

Facilitating and encouraging MLP promotes code reuse and the use of languages suitable for the application and the programmer.

C and Fortran are excellent examples of how MLP has replaced previous rivalry by cooperation between languages.

cfortran.h allows HERMES, PAW and many others to enjoy C and Fortran.

References

1. J. Bunn, "A New Query Processor for PAW", at CHEP95.
2. A. Nathaniel, "Interfacing C and Fortran", CERN COMPUTER NEWSLETTER No. 217 (July - September 1994) p.9. This is an excellent introduction to the MLX recipes required to combine C and Fortran.
3. Paul F. Dubois, Lee Busby, "Portable, Powerful Fortran Programs", *Computers in Physics*, Vol. 7, No. **1**, Jan./Feb. 1993. This includes an introduction to cfortran.h.
4. W. Wander, "DAD - Distributed ADAMO Database system at Hermes", at CHEP95. K. Ackerstaff, "A Tcl/Tk Database Interface to ADAMO and DAD", at CHEP95.

THE NEW ZEUS RUN CONTROL SYSTEM: AN EXERCISE IN MODULARITY

JAROSLAW MILEWSKI (*)

CHRISTOPHER YOUNGMAN (**)

(*) *II. Institute of Experimental Physics, Hamburg University, Luruper Chaussee 149, 22761 Hamburg, Germany. milewski@desy.de*

(**) *Deutsches Elektronen-Synchrotron DESY, Notkestrasse 85, 22603 Hamburg, Germany. youngman@desy.de*

After the first two years of data taking the ZEUS run control (RC) has been rewritten from scratch. The new system consists of loosely coupled, ZMP-interconnected processes which obey a minimum set of common protocols and standards, may migrate from one machine to another and may be easily exchanged for new modules. The new system proved to be very reliable, much more efficient, simpler, and easier to maintain.

1 Motivation

After the first two years of ZEUS data taking it became clear that the original run control (RC) system[1] had little chance to hold out till the end of the experiment. Its operation was just sufficient to take data, but apart from that it suffered from numerous problems:

- Run sequencing assumed unnecessary dependencies between participating processes, so that global state transitions were very slow and blocked in the presence of any difficulties.
- The central RC program was dealing with almost everything: run bookkeeping, producing fat run summaries, and, worst of all, collecting all kinds of status information from the components and serving it in turn to many other processes.
- The underlying code was strongly dependent on FORTRAN, VMS and DECnet, with added-on UNIX and TCP/IP patches.

All this made the system over-complicated, highly centralised, non-portable and at the end almost unmaintainable.

To overcome those difficulties, the ZEUS Central Data Acquisition Group have implemented a brand new ZEUS run control system during the 94/95 winter shutdown.

2 Prerequisites

The new RC facility had to observe the general line of the ZEUS online environment evolution, founded, among others, on the following principles:

- *Interconnect uniformly the heterogeneous component subsystems running on different types of machines.*
  In ZEUS terms it means passing XDR-encoded data over the ZMP message-passing system[2]. This is the minimum requirement on which higher-level protocols are built.
- *Apply clear distinctions of competences among the central and peripheral online tasks.*
  For instance, the new central RC facility deals with the run control and only with the run control. Such issues as safety control or data quality monitoring are put totally outside the scope of the RC system, although RC provides its own portion of information for those facilities whose competences do cover safety or data quality monitoring.
- *Every piece of information should be available at source.*
  This is another variation of the clear competences rule, this time applied to the ZEUS public data services. The central RC task provides only 'genuine' RC data like the run number, trigger type, participating components, etc. If someone is interested e.g. in the components state, he should address the components.
  (More about the ZEUS online software philosophy in another paper in this volume[3]).

One important factor which has always been present behind the scenes is the diversity of ZEUS component systems. For historical reasons, they were developed independently so that any central task of the ZEUS online has to assume very little about their similarities. Also in case of the central RC the common denominator of component message formats and even of their state transition logic is pretty small. The ways the new run control system can cope with this heterogeneity will be pointed out when necessary.

3 System overview

The architecture of the new ZEUS run control system is shown in Fig. 1. Shaded items belong to the proper RC system, the remaining ones are selected cooperating processes.

3.1 The manager task (RCM)

The Run Control Manager is the centre of communication of the RC system. It is also responsible for

- global state transitions and run sequencing,
- processing of the trigger configuration files,
- bookkeeping of the run number, shift crew, etc.

It is worth mentioning that the new manager task evaluates the global state of the system and also performs the global state transitions in *a totally different way*. Nevertheless, the new logic describes the system consistently, as ZEUS is still taking data!

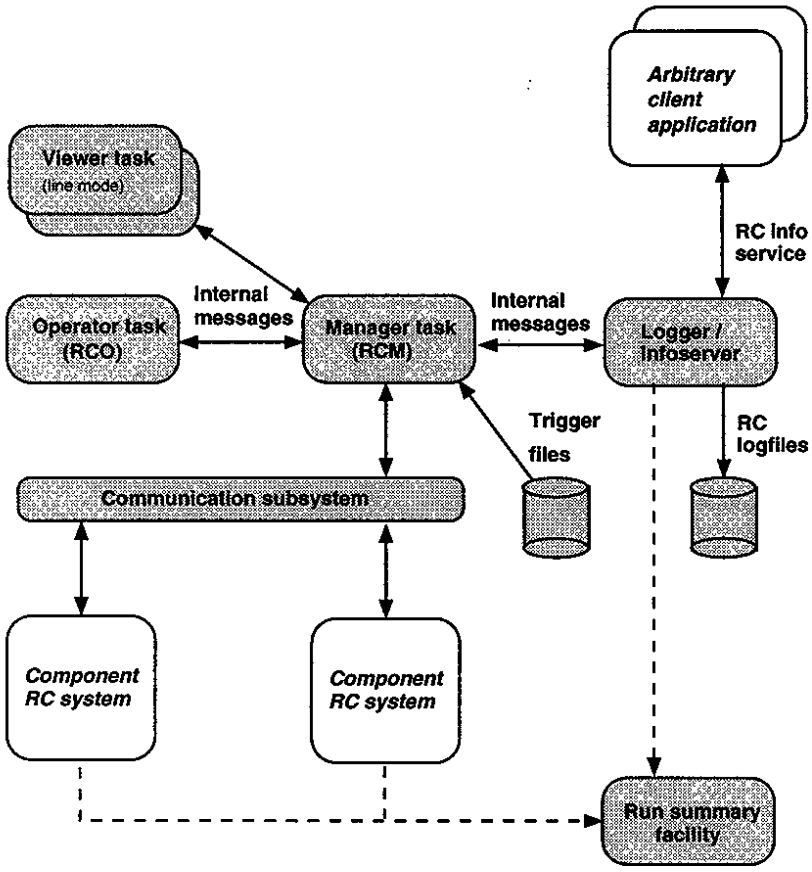

Figure 1: The ZEUS RC system architecture

Not only taking data, but doing it remarkably better than before. The average global transition time dropped down by a factor of 3 to 4, depending on trigger type and participating components. The total turnaround of the sequence "end the run" + "setup for the next run" + "start data taking" went down from 8-10 minutes with the old RC system to 2-3 minutes with the new one.

The improvement is due mainly to:

- Better run sequencing. RCM performs global transitions in *parallel steps* instead of the previous strict sequence.
- Efficient communication of RCM with the components.
- Improvements on the components side, motivated by the central RC upgrade.

3.2 Processing of the trigger files

The ZEUS 'trigger files' define the sequences of messages which are to be sent to individual components during setting up for the next run.

Due to the diversity of the ZEUS components the trigger files actually contain arbitrary strings, which are passed to the component systems in the prescribed sequence, but without any additional interpretation by the central RC program. The new RC system uses the *C preprocessor* in order to take advantage of the similarities between the existing ca. 70 trigger files. The possibility to #include common fragments of trigger definitions, to parametrise them by #defined constants, and to introduce conditional definitions on #if-#then-#else basis turned out to be surprisingly flexible.

3.3 The operator's task (RCO)

The operator's interface to the run control system works as a separate program (RCO) connected to RCM via ZMP. The ZMP name server assures the presence of only one instance of RCO running at a given time. Of course, RCO normally runs on one of the workstations in the ZEUS control room.

3.4 The viewer tasks

They use the same protocol as RCO for obtaining the continuous 'screen update' data from RCM. However, in contrast to RCO,

- The ZMP name server allows an arbitrary number of viewers to be active,
- The viewer has no possibility to issue the run control commands,
- Spying on the situation in the ZEUS control room from remote locations is easier when the viewer provides a terminal-type interface instead of an X interface.

3.5 The logger/infoserver task (RCX)

RCX is essentially an extension of RCM, which frees the main task from dealing with RC logfiles and serving of RC-related data to external clients.

The main reason for separating the two is that RCM functionality is necessary for data taking, whereas the availability of logfiles or of a public data service is a (very useful) option. If so, any possible error related to those extra functions (e.g. an error writing the logfile, thousands of requests sent to the data server by an undisciplined client) have no influence on the operation of the manager task.

RCX obtains the RCM data stream in exactly the same way as RCO or other viewers. One more thing it provides is the RCM portion of the run summary. This is neither RCM nor RCX, though, that produces the *global* run summary.

3.6 Run summary production

The run summary is a short document which describes the most important characteristics of a completed run, like the run configuration (number and type, trigger configuration, partici-

pating components), the amount, average speed, etc. of data taking, plus any summary information that the individual components find relevant.

The old run control program was producing the entire run summary itself, basing on the information collected from the components during the run. The new facility just merges arbitrary run summary files provided by the components, taking into account only the synchronisation issues (the components produce their pieces at different speeds and at different moments of time after the run completion). The absence of any part, including the RCM part, is signalled but tolerated.

3.7 The communication subsystem

At present, only half of 22 ZEUS RC components use the new communication system (ZMP) and new message formats for direct communication with RCM. The other ones pretend to talk to the old RC with message formats and communication functions (ZIP) unchanged. The reason is that components were not required to move to the new standard during the same winter shutdown period as the RC system emerged, due to time constrains.

In order to tolerate the mixture of old and new style components, ZIP/ZMP and ZMP/ZIP gateways have been placed between the old style components and RCM; the latter uses only ZMP. It is expected that most of the RC components will ultimately move to the new standard during the 95/96 winter shutdown. From the central RC point of view, the existing gateway processes may be taken out of the system and replaced by direct ZMP communication at any time.

4 The development cycle

The full development of the 100% new run control system took 4 months (September-December 1994), including 2 weeks for operator's interface. The tools used included C++ (the central run control program), XDR (all protocol specifications) and Tcl/Tk (graphical user interfaces).

The system went into operation as a fully mature product and runs very stably from the beginning up to now. Only a few bugs were spotted and relatively few improvements were found to be necessary.

The use of ZMP has been one of the key factors in successful testing and maintenance. The ZMP name server frees the run control processes from being assigned to fixed machines, which enables fully transparent migration of all the central RC tasks from one machine to another in case of any hardware or system problems. The logical division of the ZMP name space into separate domains allows 'virtual' yet complete run control environments to coexist with the proper RC system. This helped considerably in the smooth reintegration of the ZEUS components with the new run control and in the further development of new features.

5 Conclusions

Since we promoted the term "modularity" in the title of our paper, let us summarise what kind of modularity we were trying to achieve in the design of the new ZEUS run control system.

Although fully-fledged object-oriented programming was applied in the production of the central RCM program, this is not the modularity at the level of objects building up a program that was discussed here. We were rather interested in modularity at the architectural level, i.e. at the level of tasks building up the complete system.

Now, one way of speaking about the modularity in system architecture is the 'vertical' modularity of software layers, covering each other conceptually and functionally until the level of application is reached. Although the new ZEUS system is also based on layered design, mainly on communication layers built upon ZMP, this is also not the kind of modularity we would like to highlight.

The idea we are trying to promote here is something one could call the 'brick' modularity, i.e. the method of building the system out of very loosely coupled parts which observe a really minimal set of protocols and standards, whereas not the reusability of parts is most important, but their exchangeability.

In case of the ZEUS run control system nothing happens if a component changes the format or contents of its setup or run summary data, and very little happens if the component changes the format or contents of the messages sent to RCM. The central run control program doesn't care about either the location or format of its own logfiles, nor about the functionality of its subsidiary data server and cares very little about the presence of the operator or viewer tasks. The communication of RCM with the components is loose enough to permit the restart of all the central RC programs at any time, even in the middle of data taking.

Certainly, only all the parts together make a fully functional system and normally all of them are up and running. Nevertheless, the principle of minimum interdepency makes the system more robust, easier to understand, to maintain and to develop.

Acknowledgment

The work of J. Milewski is supported by BMBF grant 05-6HH29I.

References

1. I. Park, *Run control system of ZEUS online data acquisition,* in: *ZEUS contributed papers of CHEP'92,* DESY report 92-150, Hamburg (1992), pp. 43-46.
2. J. Milewski, C. Youngman and A. Kotanski, *The ZEUS message-passing system,* proc. CHEP'94, San Francisco (1994), pp. 177-181.
3. J. Milewski and C. Youngman, *ZEUS online environment on its way to open, distributed computing,* CHEP'95 (in this volume).

ZEUS ONLINE ENVIRONMENT ON ITS WAY TO OPEN, DISTRIBUTED COMPUTING

JAROSLAW MILEWSKI (*)

CHRISTOPHER YOUNGMAN (**)

(*) *II. Institute of Experimental Physics, Hamburg University, Luruper Chaussee 149, 22761 Hamburg, Germany.*
milewski@desy.de

(**) *Deutsches Elektronen-Synchrotron DESY, Notkestrasse 85, 22603 Hamburg, Germany.*
youngman@desy.de

We point out some of the difficulties encountered by the ZEUS online software in the initial phase of its operation and then propose a remedy to those problems. The general guidelines of the present ZEUS online evolution are summarised and examples of software development following those guidelines are given.

1 Introduction

When we speak about Computing for the Next Millennium, we often get fascinated with GigaFlops of computing power, GigaBauds of network throughput or TeraBytes of storage capacity. What is equally important is how to make all those technological wonders work together. Among others, the key issue is to make the systems

- more open, so that the heterogeneity of computing resources does not prevent us from gaining the best of each of them,
- more distributed, so that the cooperating resources do not disturb each other in the work on the common goal.

The ZEUS online environment is an interesting example of a complex system evolving in the direction of more open and more distributed computing. After the first few years of ZEUS operation (including 2 years of 'real' data taking) we have observed several problems in the original design of the ZEUS control software:

- Communication based on central message switching (such as CERN OSP[1]) suffered from bottleneck and single-point-of-failure problems.
- Ad hoc solutions based on direct TCP, UDP or DECnet communication did not help; they were leading to a maze of hidden interconnections between tasks.
- Heavy-weight central facilities like the central Slow Control or the central Run Control ended up by serving all possible pieces of status/error/data information to everybody, departing from their genuine control functions.

- The ever growing variety of 32-bit and 64-bit, VMS, UNIX-like and OS-9 based machines caused data exchange and process migration problems.

In this situation the ZEUS Central Data Acquisition Group took several decisions aiming at a radical change of the ZEUS online environment.

2 The guidelines

Let us formulate some of the general rules and guidelines that govern the present evolution of the ZEUS online software.

2.1 Communication as the basis

Probably 90% of online software of any HEP experiment is directly connected with moving the data around. Therefore,

> *Selecting the right communication system is possibly the most important decision in the experiment's online software.*

The communication system must, of course, be available on all the machine types used by the experiment and support platform-independent representation of data. In order to reduce the inter-process dependencies we have further decided to use asynchronous and connection-less communication protocols (see 3.1).

2.2 Portability of code

Component systems are often attached to dedicated pieces of hardware which can be controlled only by certain hosts. But even then the components should count with moving to other machine type or operating system at any time.

The central facilities are usually less hardware-oriented and should work on any of the major computer types used by the experiment. All the programs discussed in section 3 are capable of running under both VMS and UNIX (in its ULTRIX, OSF/1 and IRIX variants), or at least the expected time to port is no longer than a few days.

2.3 Distributed responsibility

ZEUS experience shows that best results are obtained when the functions performed by the programs are well separated and they strictly follow the competences of the people who are maintaining them.

> *Programs should do only what no other programs around do, and nothing more.*

When we apply this rule to the central online tasks, they turn out to be quite simple. For instance, the central Slow Control does not have to maintain the register of all the out-of-limit conditions reported by the components if every component can resend all the warnings on request. The central Run Control cares only about those aspects of component states which are relevant for run sequencing, etc.

2.4 Distributed data sources

Distributed responsibility means also that no program should serve data actually produced by somebody else.

Every piece of information should be available at its very source.

For instance, the momentary luminosity values are to be provided by the ZEUS LUMI component and not by the central Slow Control, the Global First Level Trigger rates should be made public by the GFLT and not by the central Run Control, and so on.

2.5 Distributed functionality

Instead of having a complex program performing a complex task, split it into a set of simple programs, each of them covering a well-defined subset of the required functionality.

Note that the whole set is not simpler than the original program: the communication layer may even add a new dimension to complexity. The whole set is also not necessarily easier to maintain: the maintenance effort is just more distributed. But what is important for us is that every single piece may be easily changed without affecting the other pieces if only the communication standards are observed.

3 Software developments

This section presents some of the software projects carried out by the ZEUS Central Data Acquisition Group (CDAQ) in the past 1.5 years. All of them follow the guidelines sketched in the previous section.

3.1 Communication

The ZMP message passing system[2,3] has been developed as a uniform computing platform for all the ZEUS online tasks. It is a fully distributed communication package, which may be seen as an asynchronous alternative to RPC, designed especially for real-time control tasks. ZMP relies on platform-independent data representation standard (XDR) to ease data exchange. Moreover, ZMP uses a dynamic name service which frees the online processes from being attached to definite machines.

3.2 Data servers

We have started to free the central Slow Control and the central Run Control facilities from serving all possible kinds of information to everybody. Instead of that, numerous ZMP-based data servers attached to triggers, data taking components, etc. have emerged as a replacement for overloaded data services of the central facilities. In that way the data are available directly at the place where they are really produced and there are less doubts who is responsible for proving what type of information to the rest of the experiment.

Apart from the ZEUS data servers, there exists a sound server that emits audio messages for the shift crew, gateway servers importing data from other experiments at DESY, etc.

3.3 Cross-platform process monitoring

A preliminary version of a general-purpose process monitoring and restart system has been released. In case of any vital process crash a replacement is started automatically on any suitable machine, in cooperation with the ZMP name service.

The 'restart daemons' running on different types of machines are driven by configuration files which describe the criteria and methods for restarting (and killing, if needed) of the processes under control. What is still missing are more elaborated algorithms taking into account process groups and process relationships.

3.4 Cross-platform symbol definitions

The ZEUS online as a whole maintains implicitly a number of 'environmental variables' like the name of the printer where some summary report has to appear or the number of the last run. Since those variables must be shared by processes running on different machines under different operating systems, a simple dictionary-type server provides the variable bindings to all the interested parties. Due to a destructive read operation on symbols, it can also be used for synchronisation of processes running on different machines.

3.5 The new Run Control system

The central ZEUS Run Control system has been rewritten from scratch during the last winter shutdown, following the new standards of the ZEUS online. It consists now of a set of loosely coupled processes running on different machines, so that e.g. message logging, run sequencing and run bookkeeping are almost totally independent. (The new ZEUS RC is described in more detail in another paper in this volume[4]).

An equally radical upgrade of the Slow Control system is planned for the 95/96 shutdown.

4 The final word

The reader would probably agree that the keywords 'open' and 'distributed' are closely related to modularity of software design. However, our understanding of the word 'modular' is rather specific. We are less interested in the reusability of code but rather in exchangeability of code: every brick must be ready to be thrown away and replaced by something better if needed. The smaller, simpler, and platform-independent the bricks are, the better. The only thing the components of the system really share are the communication standards.

Acknowledgment

The work of J. Milewski is supported by BMBF grant 05-6HH29I.

References

1. R. Jones, G. Mornacchi and R. Russell, *OSP user's guide, version 3.0,* CERN DD/OC Group (1990).
2. J. Milewski, C. Youngman and A. Kotanski, *The ZEUS message-passing system,* proc. CHEP'94, San Francisco (1994), pp. 177-181.
3. A. Kotanski, J. Milewski and C. Youngman, *The ZEUS message-passing system (ZMP), release 1.2,* ZEUS note 94-150, DESY Hamburg (1994), 59 pp. *http://zow00.desy.de:8000/ZEUS_MANUALS/ZMP_spec.ps*
4. J. Milewski and C. Youngman, *The new ZEUS run control system: an exercise in modularity,* CHEP'95 (in this volume).

The ZEUS Central Tracking Detector Second Level Trigger

H. A. J. R. Uijterwaal, A. Quadt, S. Topp-Jorgensen and R. C. E. Devenish

University of Oxford, Department of Physics, Keble Road, Oxford, UK
ZEUS Collaboration

This paper describes the Second Level Trigger for the Central Tracking Detector (CTD-SLT) of the ZEUS experiment. The CTD-SLT consists of a network of microprocessors running a track-finding algorithm. Operational experience gained during the 1995 HERA data taking period shows that the maximum processing rate approaches 800 Hz, with almost 100% track finding efficiency and adequate p_T and vertex resolution.

1 Introduction.

In the HERA accelerator[1] protons with an energy of 820 GeV collide with positrons of 27.5 GeV. The bunch-crossing rate is 10.4 MHz. The ZEUS detector[2] is one of the two all-purpose detectors for HERA. Its main elements are an inner tracking system, consisting of a forward, central and rear tracking detector surrounded by a superconducting coil providing a magnetic field of 1.43 T, a high resolution calorimeter, muon detectors and several special purpose detectors.

ZEUS uses a 3-level trigger system to cope with the high interaction rate of HERA. The first two levels each consist of a trigger system per component and a global box. The component triggers extract information from a single subsystem. The results are sent to a global box, where the data from all subsystems is combined into a global decision. The eventbuilder (EVB) then collects the data from all components for the third level trigger (TLT). The TLT consists of a computer farm running a subset of the off-line reconstruction code on the full event data.

The background rate at HERA is of the order of 50 to 100 kHz, which is high compared to the rate of interesting physics interactions of at most 10 Hz. The trigger system should reject the backgrounds while at the same time maintain a high efficiency for physics interactions. During the 1995 data taking period, the typical output rates of the 3 trigger levels are 350, 50 and 10 Hz respectively, although later this year FLT rates of up to 700 Hz are expected.

2 The ZEUS Central Tracking Detector

The ZEUS Central Tracking Detector (CTD) is a cylindrical wire chamber with an inner and outer radius of 16 and 85 cm respectively. It is located along the beam line between $z = -100$ and $z = +105$ cm when measured from the nominal interaction point. It covers a polar angle (θ) from 15° to 164° and the full azimuthal angle (ϕ).

The CTD consists of nine cylindrical "superlayers", each consisting of 32 to 96 cells. The cells have field wires and 8 sense wires. In the odd numbered, or axial, superlayers, the wires run parallel to the beam-line. In the other superlayers, they have a tilt of ±4°. The sense-wires in the CTD are read out by FADC's, giving hits in (r, ϕ). In addition, the 3 inner axial superlayers are read out by a

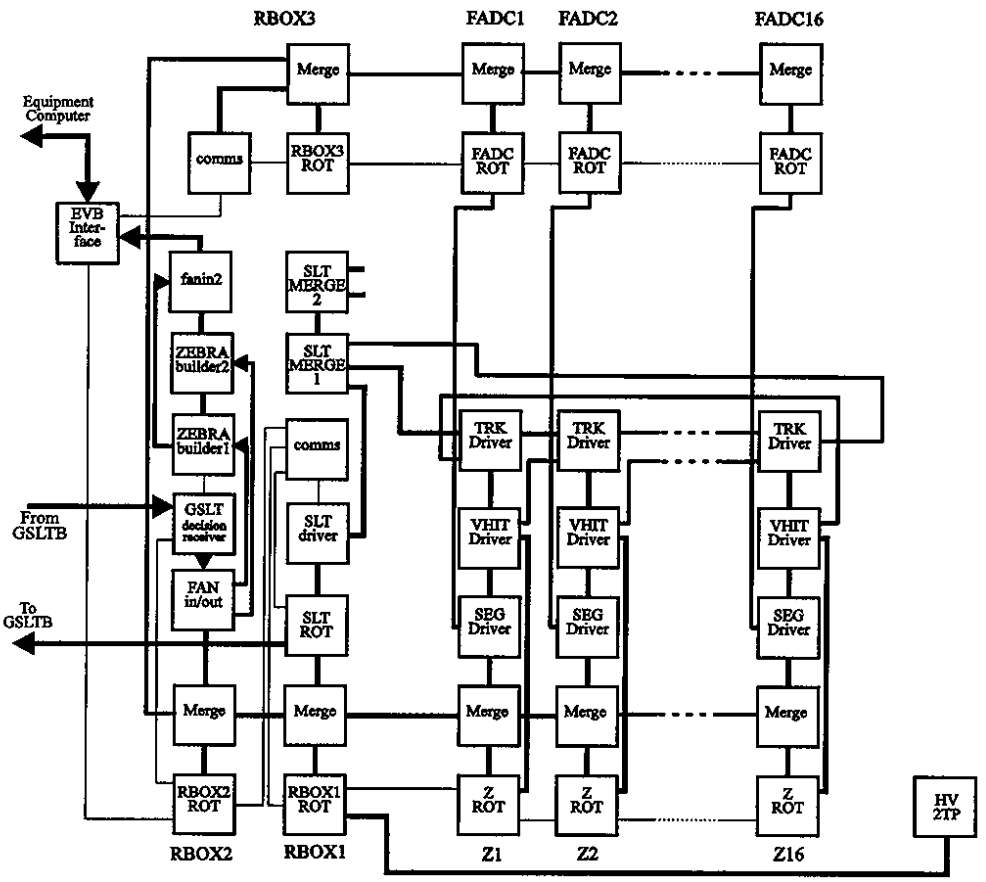

Figure 1: The CTD SLT and Readout Transputer Network.

z-by-timing ($z$bt) system, giving hits in (r, ϕ, z). The resolution of the hits is about $\sigma_{\mathrm{FADC}} \sim 190\ \mu$m and $\sigma_{z\mathrm{bt}} \sim 4$ cm.

3 The CTD Second Level Trigger

The CTD has a first and second level trigger system (FLT, SLT). The FLT uses information from the z-by-timing system only to look for tracks in pre-defined masks.

The goal of the CTD-SLT is to find tracks in the chamber, using data from both the z-by-timing and FADC systems, and combine these tracks into an event vertex. The event vertex can be used, together with time-of-flight information from the calorimeters, to distinguish between interactions coming from the e^+p interaction region and beam-gas backgrounds. In addition, the individual tracks can be used to detect more specific signatures of certain classes of physics events.

4 The CTD-SLT Trigger Algorithm.

The CTD-SLT trigger algorithm uses a 4 step approach in order to find tracks: first track-segments in (r, ϕ) are found. These track segments are then transformed into vectors using information from the z-by-timing system. The vectors are combined into tracks and finally all tracks together are used to determine the event vertex.

The starting point for the track-finding are the (r, ϕ) hits from the FADC system in each cell of the CTD. The algorithm takes a straight line determined by 2 seed-hits and looks for hits on neighbouring wires within a "road". If 3 or more hits lie within the road, the line is called a segment and the calculation is repeated for the other hits in the cell.

The second step assigns hits in a cell with the same drift time in both the FADC and z-by-timing systems to the segments. The segments are then transformed into a vector in (r, ϕ, z) with a direction χ (the direction of the vector with respect to the radius vector from the nominal interaction point).

The track finding step loops over all vector hits, starting with the vector hits in the outermost superlayer and working its way inwards. For each vector hit (the "seed"), the algorithm loops over the vector hits in the superlayers inside the seed layer. If χ and ϕ of a vector hit are consistent with a track originating from $r = 0$ in the direction of the seed, then the vector hit is assigned to the candidate track. If at least 3 vector hits, or at least 2 vector hits from the inner 3 superlayers, can be assigned to a candidate track, then the track is accepted and the process is repeated for the remaining vector hits.

For all tracks, the algorithm determines p_T of the track by fitting the coordinates of the vector hits to a circle which passes the nominal interaction point. It also determines the z-origin of the track at $r = 0$ by fitting a straight line to the individual (r, z) hits. A binning algorithm is used to determine the event vertex: the z-coordinates of the tracks are binned in 20 cm wide, overlapping bins. The event vertex is the center of the bin with the most entries closest to $z = 0$.

5 Implementation of the trigger.

The CTD-SLT and the readout of the chamber are closely coupled: first the readout has to extract the data from the electronics and store it in buffer memory, then the SLT accesses the data and, in case of a positive decision, the readout has to take the data from the buffers and ship it to the EVB. For this reason, the two systems have been combined.

The algorithm described above has a large degree of parallelism built into it: finding of segments and vector-hits only requires input from a single cell, thus one processor can do the segment or vector-hit finding in a cell while a second processor at the same time deals with another cell. Moreover, the track finding stage only requires the vector hits from a wedge in (r, ϕ), two or more processors can therefore do the track-finding in certain regions of the chamber. This intrinsic parallelism is used in the hardware implementation of the trigger.

5.1 *The Transputer system.*

The processor used for the CTD-SLT and readout is the INMOS T800 transputer[3] (TP). The TP offers hardware support for parallel processing and is therefore well suited for an application with a large degree of parallelism in it.

For readout and triggering purposes, the CTD is divided into 16 sectors in ϕ. Each sector has 2 crates with electronics, one for the FADC system and one for the z-by-timing system. Two TP's in each crate take care of the readout of the electronics and the shipping of the data to the EVB.

Each sector also has 3 TP's for the SLT: one for segment finding, one for vector hit finding and one for track finding. The last TP receives input from the vector hit finding TP in this sector and its neighbouring sector. It thus covers $\frac{1}{8}$ of the CTD. Since there are 16 TP's doing track finding, each track can be found twice and the duplicate tracks have to be removed later on, but on the other hand, this allows one to find tracks with a p_T of down to ≈ 150 MeV.

The 7 TP's in each sector are part of a network of 129 TP's (see figure 1). The other TP's determine the event vertex, take care of communication with the EVB, and control the system. The TP-network is connected to a VAX workstation. This workstation is used to store files, boot the TP's and communicate with ZEUS Run Control. No major hardware problems have occurred in the TP-network since data-taking started in 1992.

5.2 *Software implementation.*

The algorithm is coded into OCCAM, the native language of the TP, and also in FORTRAN. The FORTRAN version of the algorithm is used in Monte Carlo event simulation and to study new trigger configurations for ZEUS. The two versions of the algorithm agree on an event-by-event basis at the 99% level.

The system offers ample possibilities for debugging and on-line monitoring. First of all, it is possible to feed events from previous runs into the system. This allows one to check the results or measure the rate with which events can be processed. Then the internal transputer clock is used to measure the time used in the various processing steps. This gives a indication of the performance of the system during data-taking. Finally, all results of the algorithm are available off-line and can be compared against the simulation or off-line reconstruction.

5.3 *Rates and data volumes.*

The system has been designed such that all processing and data-transport steps take, on average, 1 ms thus giving a throughput of 1 kHz. However, the number of hits in the chamber shows large fluctuations and it can take up to 50 ms to process the extreme events. In order to cope with these fluctuations, each TP in the system has input and output buffers and the system will continue to run under all conditions.

However, a latency of up to 50 ms is not acceptable for other components in ZEUS. In order to limit the maximum latency to about 20 ms, the number of segments that can be found in a sector is limited to 6 and the number of tracks that can be found to 20.

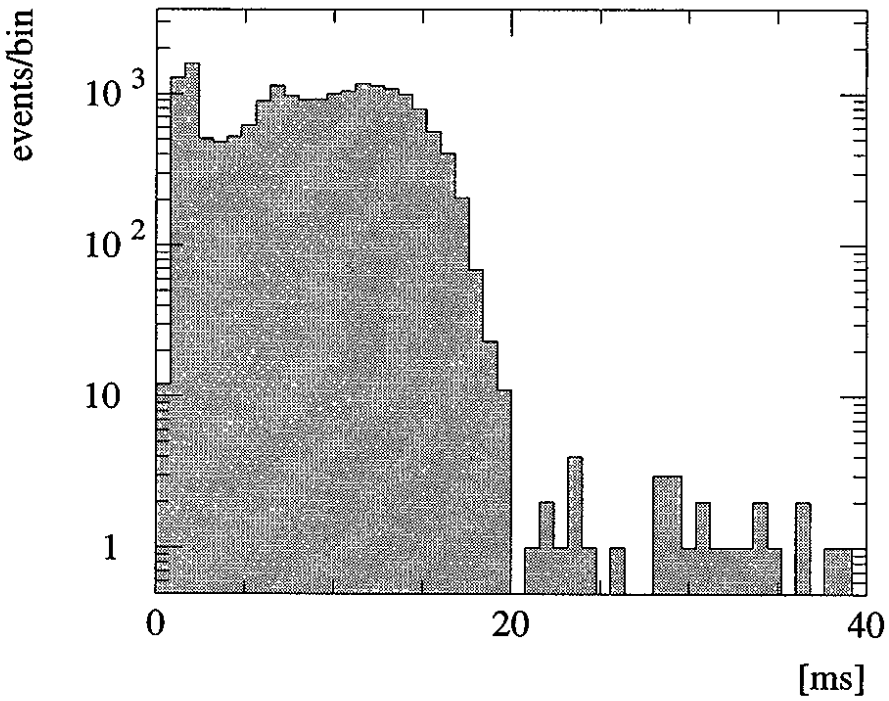

Figure 2: Latency of the CTD-SLT during data-taking.

6 Performance of the CTD-SLT.

The performance of the system has been measured by feeding events with known characteristics into the readout-buffers and processing them. Extrapolating from these measurements gives an estimate of 750 Hz as the maximum processing rate. The actual maximum processing rate during physics data-taking has not been measured yet, due to the lower than expected luminosity. However, during a test run with lower trigger thresholds, a maximum rate of 800 Hz was obtained, albeit with a high dead-time of 25%. The latency of the system is shown in figure 2. The shape of this curve is consistent with performance measurements of the readout (0 to 5 ms/event), the processing time on each TP ($\approx$ 0.75 ms on average) and the time needed to ship the data inside the system (1 to 10 ms/event).

The single track finding efficiency and p_T-resolution of the CTD-SLT have been studied on Monte Carlo events with a single track. From these studies, it is found that the track-finding efficiency is $(98 \pm 1)\%$ for tracks passing through the inner 3 superlayers of the chamber. The p_T resolution for these tracks (see figure 3) is $\sigma_{p_T} \sim 0.04 p_T^2$ (p_T in GeV), compared to $\sigma_{p_T} \sim 0.005 p_T^2$ for full off-line reconstruction. Figure 3 also shows the vertex resolution of the events compared to off-line analysis. The resolution is about 9 cm, time-of-flight methods give a resolution of 20 cm.

The resolutions obtained with this algorithm are sufficient for SLT purposes. In the present ZEUS trigger configuration, the event vertex provided by the CTD-SLT

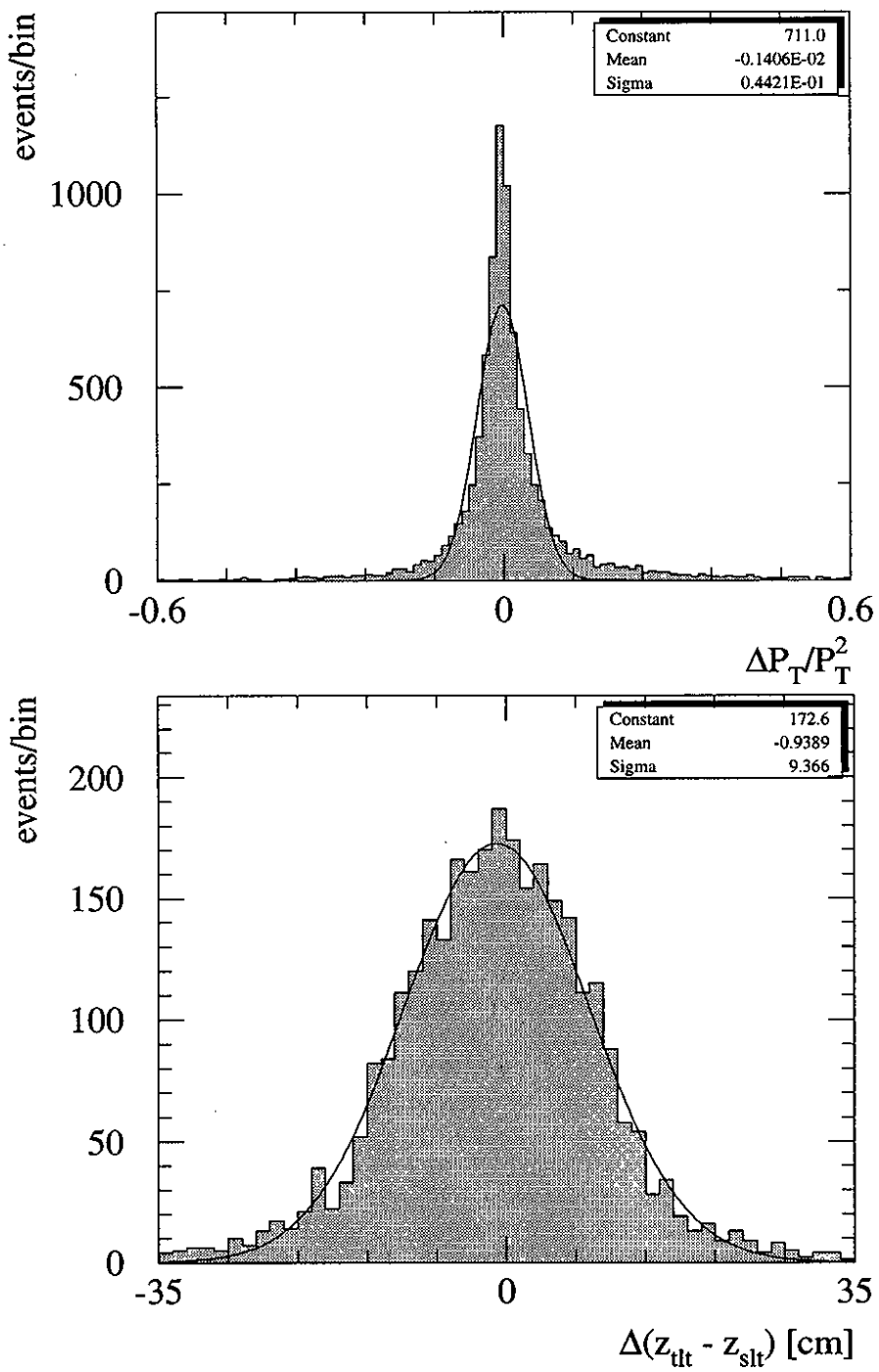

Figure 3: Top plot: p_T resolution of the algorithm (MC data). Bottom plot: Vertex resolution of the CTD-SLT (real data).

is used to separate background and signal events for various physics studies. The individual tracks are used to identify high p_T tracks from heavy quark decays.

7 Conclusion.

The results presented in the previous section show that a track-finding algorithm can be implemented on a network of parallel processors and provide reliable tracking information within the time constraints set by the high interaction rate at HERA. Comparing to the off-line analysis code, the results of this algorithm are only a factor of 10 worse, while the system processes the events about 1000 times faster. The algorithm itself can be used for other tracking chambers and as faster processors become available, a similar approach could be used for future HEP experiments.

References

1. R. Brinkmann, DESY HERA 88–03.
2. The ZEUS collaboration, M. Derrick *et al*, Phys. Lett. B293(1992)465.
3. INMOS Limited; The Transputer Databook, 2nd edition; 1989.