DEUTSCHES ELEKTRONEN – SYNCHROTRON

DESY 92–141
October 1992

ZEXP - Expert System for ZEUS Problem Analysis, Theoretical Background and First Results

U. Behrens, M. Flasiński, L. Hagge

Deutsches Elektronen-Synchrotron DESY, Hamburg

ISSN 0418-9833

NOTKESTRASSE 85 · D-2000 HAMBURG 52

To be sure that your preprints are promptly included in the
HIGH ENERGY PHYSICS INDEX,
send them to (if possible by air mail):

DESY **Bibliothek** **Notkestraße 85** **W-2000 Hamburg 52** **Germany**	**DESY-IfH** **Bibliothek** **Platanenallee 6** **O-1615 Zeuthen** **Germany**

DESY 92-141
October 1992

ISSN 0418-9833

ZEXP - Expert System for ZEUS Problem Analysis, Theoretical Background and First Results

Ulf Behrens, Mariusz Flasiński*, Lars Hagge
Deutsches Elektronen-Synchrotron, Hamburg

Abstract

The proper and timely reactions to errors occurring in the on-line data-acquisition (DAQ) system are necessary conditions of smooth data taking during the experiment runs. Since the Eventbuilder (EVB) is a central part of the ZEUS DAQ system, it is the best place for monitoring, detecting, and recognizing erroneous behaviour. ZEXP is a software tool for upgrading the DAQ system performance.The pattern recognition methodology used for designing one of its two main modules is discussed. The general design ideas of the system and some preliminary results from the summarizing run module are presented, as well.

1 Introduction

The proper and timely reactions to data throughput errors occurring in the online data-acquisition (DAQ) system are necessary conditions of smooth data taking during the experiment runs. Since an occurrence of such errors sometimes results in the loss of valuable data from the detector, software tools for upgrading the online DAQ system are needed. The ZEUS Expert System (ZEXP), currently being designed and implemented in the Eventbuilder (EVB) environment, is one of such tools [1, 2]. The EVB is a well suited place for embedding the Expert System, because it is the central data throughput process in the online DAQ system. Moreover, a lot of monitoring information related to the performance of the EVB and those components, which interface with EVB is available due to its monitoring utilities [2]. This preprocessed information is sent to the Run Control (RC). On the other hand, only the symptoms of an anomalous system behaviour are dispatched to RC. Frequently, it does not give sufficient information for taking adequate actions by RC and results in aborting the run. Therefore,

- detecting anomalous behaviour of the DAQ system,
- recognizing (classifying) anomalous behaviour,
- reasoning a way of treatment, and
- providing the synthetic knowledge about present data throughput conditions

are necessary conditions of taking the proper and timely action during a run, and setting the proper parameters for the next runs. Of course, designing a present version of ZEXP on the basis of only EVB monitoring data limits its expertizing abilities. Due to using a Structural Analysis and Structural Design technique and a prototyping method for a ZEXP design, extensions are possible (they are discussed in the final section).

The methodology assumed for designing the system and the model of the system is presented in Section 2. The theoretical foundations related to designing one of two main modules of the expert system, namely a *pattern recognition* module classifying erroneous behaviour of the on-line DAQ system, are presented in the third section. Preliminary results from an implemented process, which summarizes runs are discussed in the fourth section, whereas concluding remarks are included in the final section. The appendix contains a short description of the process implementation.

2 Problem analysis

2.1 General remarks

A requirements (problem) analysis is the crucial step in the process of a software development. DeMarco researches into a methodology of a software development [3] have shown the following distribution of software errors: poor requirements analysis (model building) - 56% , bad design - 27%, coding errors - 7%, and other - 1%. Similar researches concerning the cost

*DESY/University of Siegen, Germany. On leave from Department of Computer Science, Jagiellonian University, Cracow, Poland

of rectifying software errors have underlined even more that fact. Its distribution is : poor analysis - 82%, bad design - 13%, coding errors - 1%, and other - 4%. Therefore, a solid and careful problem analysis with the help of appropriate methods and tools is a necessary condition of producing reliable software of high performance. It seems to be even more important, if a developed software is to be embedded into an existing software system. In this case, some additional problems (e.g. interfacing, changing assumptions connected with the fact that existing software is modified/upgraded) may arise and complicate model building.

A number of methodologies developed for building models of software systems in the seventies and eighties, like the DeMarco approach [3], the Yourdon methodology [4], or the Hatley-Pirbhai method [5] are widely recognized and used by software engineers. Recently, such methodologies have been defined for analysis and design of distributed systems [6, 7, 8]. Their common feature is using diagramming techniques (usually of a graph-theoretic origin) associated with data bases containing the complete information about the system. Therefore, their use is especially recommended, if a software group building (or maintaining) a big system is not based on a permanent staff. Applying such systematic methodologies leads not only to producing reliable software, but also results in creating a detailed documentation of the system. (This means e.g. that a newcomer can gain weeks (or months) of experience just looking into a system model that has been built.) What is more, the process of software upgrading is much easier and it is possible that even people, who have not constructed the system can improve or modify it easily.

2.2 Methodology

Models of some components of the ZEUS data acquisition system (including the Eventbuilder) have been built with the help of the CADRE Teamwork 3.0. This release is generally based on the DeMarco approach [3] (with some extensions for modelling real-time systems). The DeMarco method originally has not been developed for structured analysis of real-time systems, but rather for modelling business-like systems. Since the ZEUS Eventbuilder, in which the Expert System is to be embedded, is a typical real-time system, it seems that the Hatley-Pirbhai method [5] is more suitable for our purpose. Therefore, both requirements and architecture models of the ZEUS Expert System are being built using the Hatley-Pirbhai method, and the model is only partially constructed with the help of the newest version of the CADRE

Teamwork 4.1 (which allows to build a requirements model according to the Hatley-Pirbhai method) .

Two facts, namely the strong human - computer interactions occurring in expert systems and the development of our system in the context of the existing environment have influenced the decision of mixing the SASD methodology with the so-called prototyping method [9, 10]. In its radical form, prototyping consists in developing a prototype of the system as a model. Therefore, in the early eighties prototyping was considered to be an alternative approach to a structural design. Nowadays, these methodologies are very often applied altogether [10]. In such a case prototyping is treated as a kind of an experiment with a system model being built during a requirements analysis. Thus, the initial set of requirements is defined, then it is implemented with the intention of iterative expanding. This allows one to see how the final system will service his/her needs. The approach is especially useful for solving interfacing problems before the design of the sytem is completed. Therefore, one of the first implementing works (besides the ones, the results of which are presented in the fourth section) were focused on interfacing to the system [11].

2.3 The model

Before a general model of the system model will be presented, let us make the following remarks. The so-called decision-support systems were ancestors of expert systems. (Of course, nowadays such systems are also being built.) They did not make decisions, but they provided relevant information about various aspects of expertized problems, which allowed professional experts to make decisions. The succeeding generations of such systems did not only retrieve and display data, but also performed a variety of mathematical analyses of data and they had usually capabilities of presenting information in graphic forms. At the end, first expert systems assisting people in making decisions have been constructed. They possess three features differing them from their ancestors. Firstly, their data bases are highly organized [12]. They constitute *knowledge*, not just data. The second feature distinguishing them from common knowledge-based systems is the ability of a knowledge generalization. This means that similar events are grouped together on the basis of common important features in their knowledge data bases. Lastly, expert systems are able to explain the line of reasoning that leads to their decisions [13]. (This is connected with the fact of a rather psychological

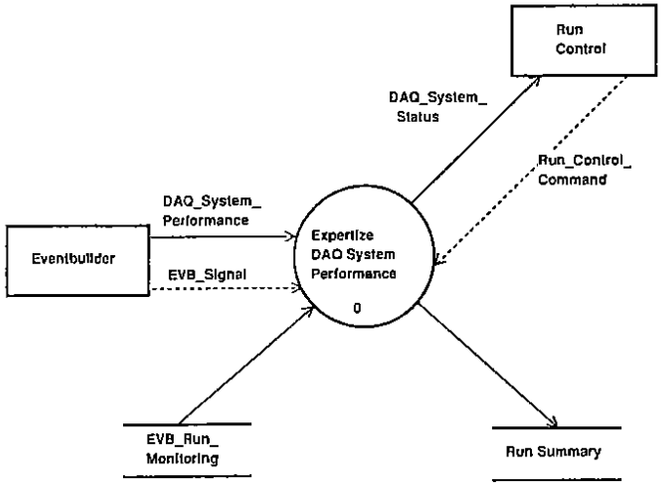

Figure 1: The Context Diagram of ZEUS Expert System

nature that people consider the system to be credible, if its behahoviour is explainable.) These features have influenced strongly the process of building the model of ZEUS Expert System.

The main purpose of the ZEUS Expert System is to provide a diagnosis and a treatment in case of anomalous data throughput of the online DAQ system. The context diagram of the system is shown in Fig. 1. The system is activated/deactivated by EVB. It takes EVB DAQ_System_Performance measurements, processes them, and sends results to Run Control. Additionally, on demand, the system produces a Run Summary, mainly of the graphics form, which contains information concerning data throughput for the whole run. This information may be used for the run analysis for setting next runs parameters as well as for the maintenance of the Expert System (modifying the system knowledge data bases).

The internal functionality of the system is shown in Fig.2. The process Detect Anomaly scans DAQ_System_Performance measurements and matches them against Erroneous_Behaviour_Templates being a kind of filters allowing the system to detect performance anomalies. The detailed analysis of results from an implemented summarizing run module (see Section 4) has shown that such anomalies can be grouped generally into two sets. The first set contains situations, which can be recognized on the stage of a detection. They are characterized by very quickly changing values of the data through-

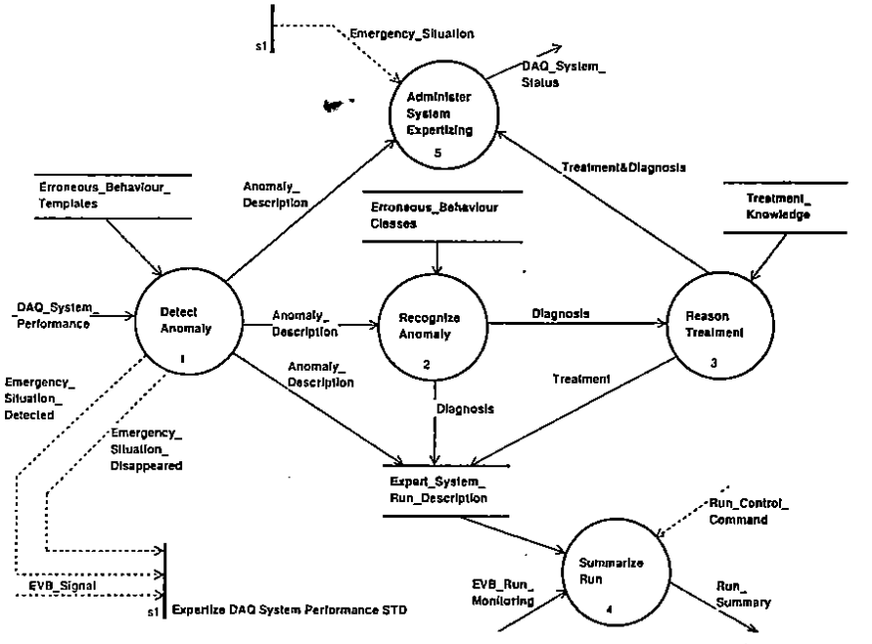

Figure 2: The Data/Control Flow Diagram of ZEUS Expert System

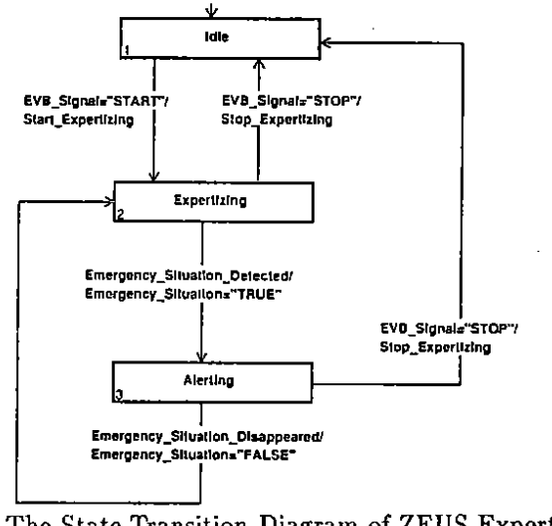

Figure 3: The State Transition Diagram of ZEUS Expert System

put parameters (in the sense of approaching the system blockage). This means that an Expert System warning should be sent to the Run Control immediately. Omitting a detailed analysis in such situations is caused also by the impossibility of obtaining additional parameters that could influence the system decision (see a discussion in Section 4). The second set includes anomalies characterized by: slower changing of parameters values and a fuzziness of a situation. In such a case, a detailed analysis is possible, and it is necessary for giving a description of the system status.

In the first case, the process Detect Anomaly sends the Emergency_Situation_Detected signal, and from this moment (till Emergency_Situation_Disappeares) it is responsible for "emergency" expertizing. (The other reason for assuming such a solution will be given after introducing the architectural model of the system.) In the second case Detect Anomaly sends an Anomaly_Description to the *pattern recognition* process (No. 2 in Fig. 2), which gives a Diagnosis for further reasoning made by the Reason Treatment process. The use of a two-step procedure (classifying/generalizing anomalies, and then expertizing a situation) is neccessary, because (as we have mentioned previously) expertizing algorithms operate on generalized (abstracted) data. The coordination of the system work (in fact, there are two decision centers activated in various situations) is the task of the Administer System Expertizing process, which reads the Emergency_Situation signal. The state transition diagram describing modes of the system work is shown in Fig. 3.

The process Summarize Run can be activated any time by the Run_Control_Command. It has been implemented firstly, because of its usefulness for analyzing the DAQ system performance and building two Expert System data bases: Erroneous_Behaviour_Classes used by the pattern recogniton module, and Treatment_Knowledge used by the expertizing module. Since both bases should contain highly-organized data that can be abstracted only with the help of a big sample set, implementing this process very quickly was very important (the first results are discussed in the fourth section). Let us notice also that the analysis of these results has influenced building the system model significantly.

The architecture model analysis was strongly determined by the fact of existing the software/hardware environment, in which the Expert System is to be embedded. The architecture context diagram is shown in Fig. 4. The functional and control processing of the system is made on an SGI 4D25. The system should take input data from one of the Eventbuilder T800 transputers, and after processing the data ought to be sent to the ZEUS Run Control implemented on a VAX 8700. No output processing is needed. (However,

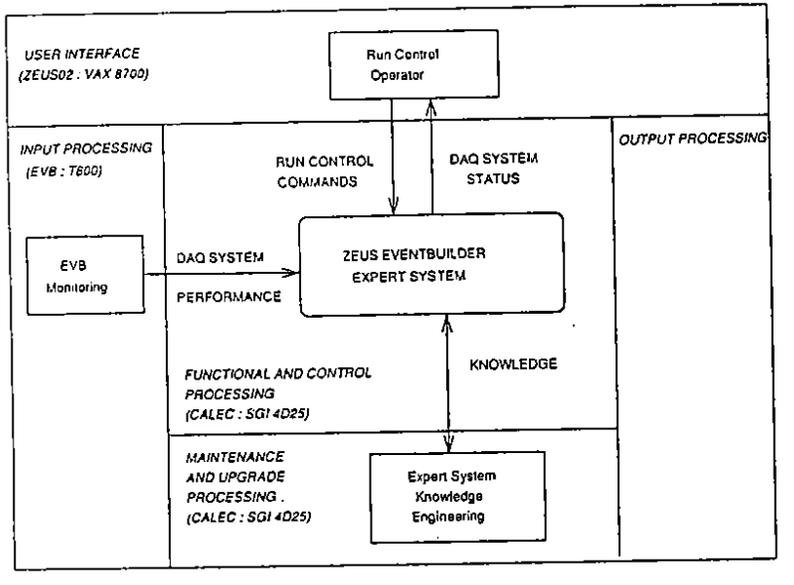

Figure 4: The Architecture Context Diagram of ZEUS Expert System

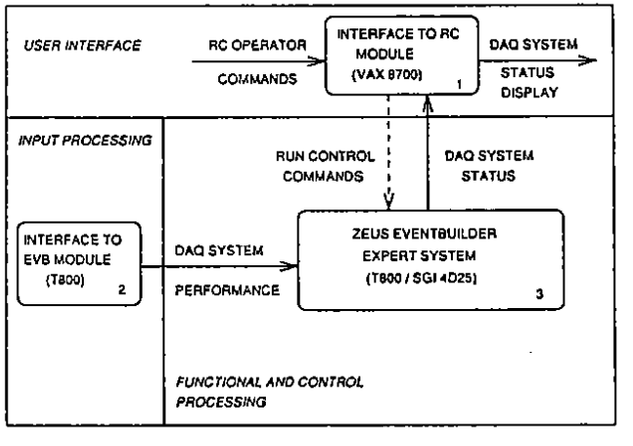

Figure 5: The Architecture Flow Diagram of ZEUS Expert System

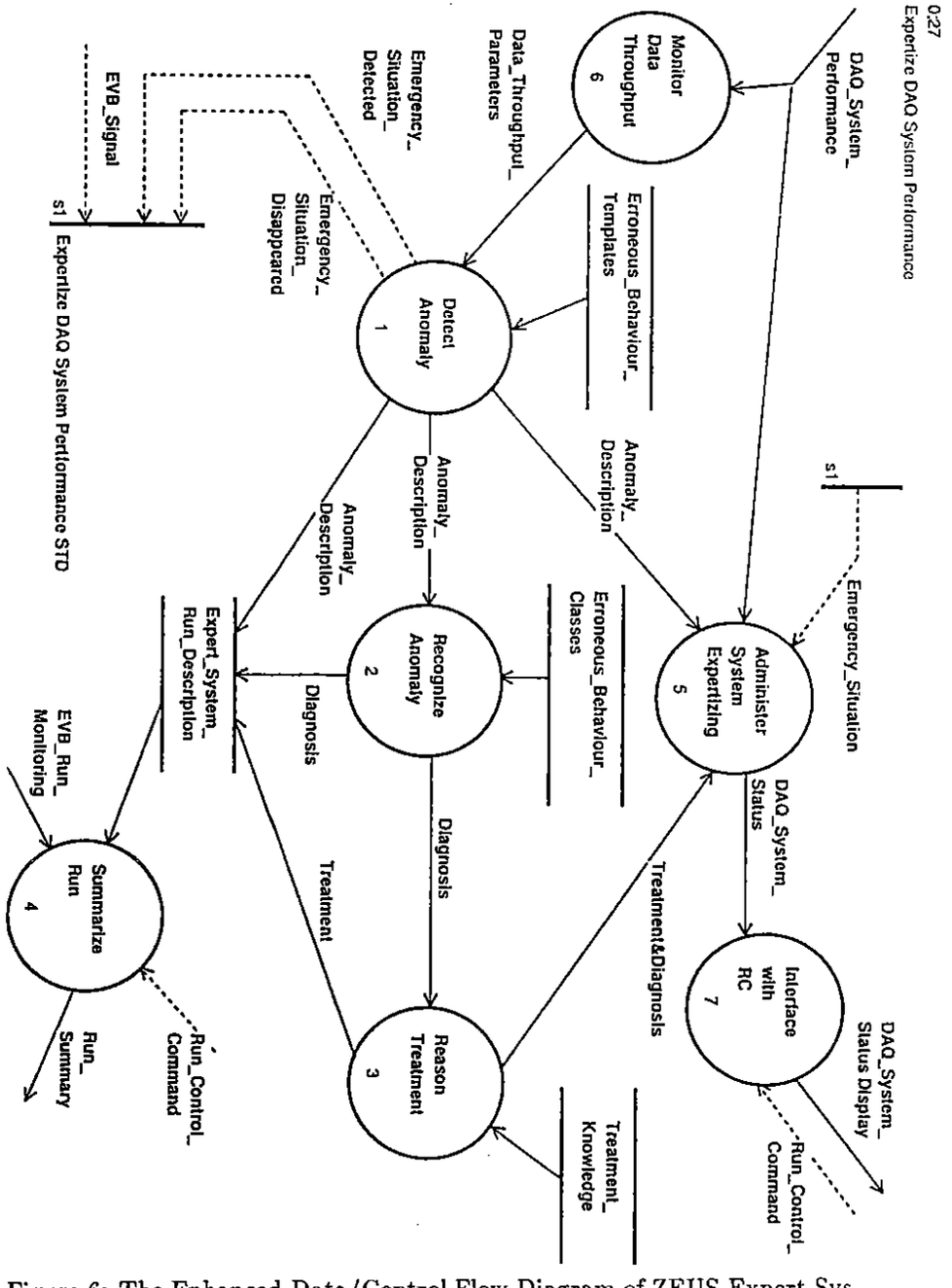

Figure 6: The Enhanced Data/Control Flow Diagram of ZEUS Expert System

in the future the system may be extended from an expertizing system to a controlling one). The maintenance and upgrading of the Expert System (mainly in the form of operations on the system knowledge data bases) will be made in the future on the SGI 4D25 as well. Embedding the Expert System in such a heterogenous hardware environment (with a previously implemented software running on it) has necessitated the implementation of additional modules interfacing with the RC and EVB systems - see Fig. 5. This has also led to the modifications of the requirements model by defining the so-called *enhanced requirements model* [5]. The *enhanced DFD/CFD*, in which the architectural requirements are taken into account is shown in Fig. 6. The process **Interface with RC** is embedded in the RC environment (on the VAX 8700) [11]. The process **Monitor Data Throughput** is located on a T800. Due to availability of this transputer **Detect Anomaly**, which is to expertize in an emergency mode is also placed here. The problem of interfacing the Expert System with the environment, and its design/implementation solution is discussed in a detailed way elsewhere [11].

3 The pattern recognition background

3.1 Preliminaries

Let $\{1,\ldots,L\}$ are *pattern classes*, into which we want to classify patterns. A pattern is defined as a vector $\mathbf{x} = (x_1,\ldots,x_N)$, where x_n, $n = 1,\ldots,N$ is a value of its nth feature. We treat a pattern as a point in an N-dimensional *feature space*. We assume that pattern classes create *clusters* in feature space. The placement of clusters is unknown, but at least the finite set of *labelled samples*

$$S = \{(\mathbf{x}^m,\omega^m)\ ,\ m = 1,\ldots,M\ ,\omega^m \in \{1,\ldots,L\}\}$$

called the *training set* is available. The aim of pattern recognition is defining a *decision rule* $c(\mathbf{x})$, such that

$$c(\mathbf{x}) = m, \text{ if } \mathbf{x} \in \omega^m \text{ for any } \mathbf{x}.$$

This is usually made by defining L *discriminant functions* $g_1,\ldots,g_L$, such that

$$c(\mathbf{x}) = m, \text{ if } g_m(\mathbf{x}) > g_i(\mathbf{x}) \text{ for all } i \neq m.$$

The methodologies used in decision-theoretic (non-syntactic) pattern recognition may be divided into nonparametric [15, 16, 17, 18], parametric (Bayes) [19, 20], and sequential [21, 22]. Nonparametric methods are used for dividing a feature space on the basis of a training set. Using parametric methods demands additional information concerning parametrized distributions for classes. Sequential methods, based on Wald's sequential probability ratio test (SPRT), allows one to take a decision before computing all the features.

3.2 Construction of the feature space

Definition of a feature space is the most critical stage of a pattern recognition process. Its aim is to transform a *measurement space* (all available raw data describing recognized phenomena) into a feature space, which

- is of a relatively low dimension,
- includes sufficient information for a correct recognition,
- allows one to use a distance as a measure of dissimilarity between classes and similarity within classes,
- allows one to express each feature consistently over all patterns.

On the other hand, the part of the pattern recognition theory concerning feature extraction is not so developed, as the one relating to pattern classification, because factors deciding whether a feature is important depend strongly on the problem considered. Therefore, in many cases one must use rather some methodological hints and adapt them for solving a particular problem, than use ready-made methods. Formally, if a measurement space is denoted by Y, and a feature space by X, then this aim is defined as finding a transformation T mapping Y into X , such that:

$$\mathbf{x} = \mathrm{T}(\mathbf{y}) \text{ , where :}$$

$\mathbf{x} = (x_1, \ldots, x_N)$,
$x_n = T_n(y_1, \ldots, y_C),\ n = 1, \ldots, N$, and
$N < C$.

The feature selection techniques can be divided into indirect methods and direct ones. A feature selection with indirect methods consists of choosing a family of parametrized transformations altogether with a certain criterion function (allowing one to measure the quality of the transformed points

distribution) and defining an optimization algorithm finding the optimum transformation (in the sense of a criterion assumed). The simple linear transformations are used for: normalization, reduction of a dimensionality of a feature space (selecting the most valuable features), and obtaining the orthonormality of a feature space (which is a necessary condition of a proper performance of classification algorithms). In complex cases, nonlinear transformations are used as a model, but in practice they are replaced by piecewise linear transformations (linear over subspaces of a measurement space Y), which can be computed more efficiently. The typical criterion functions are constructed to maximize the distances between transformed samples of different classes (the co-called interset distances) and minimizing a scatter of each cluster created after a transformation (the so-called intraset distance). Other quality measures are defined on the basis of the *a priori* class probabilities and the conditional probability densities of the transformed samples (e.g. the divergence, the Bhattacharyya distance, the Patrick-Fisher measure).

The direct feature selection methods are based on the *principle of decomposition*, stating that any problem can be decomposed into a set of simpler problems, and that each of these subproblems may be decomposed, etc. This means that well-known classification algorithms may be used as recognizers of separated particular aspects of the whole problem defining features directly. Such a defined feature gives a measure of some characteristic of a considered fenomenon. There are two approaches of constructing a new feature. The first one consists in generating a feature of the form of a discriminant function measuring continuously the degree of the measurement quality:

$$x_i = g(\mathbf{y}).$$

In the second approach, we construct subclasses in the subproblem and then we use their labels for measuring a considered characteristic, i.e.

$$x_i = m \text{ , if } \mathbf{y} \in \omega^m.$$

Direct feature selection methods are especially preferred, when the sequential approach is to be used during a classification stage. In this case a detailed analysis should be made not only on selecting features but also on *ordering* them.

Apart from transformations mentioned above, the so-called *scale conversions* play an important role in establishing features. The problem of scale

conversion arises from the fact that measurements are sometimes of varying forms or we would like to transform some measurements to other forms. The research work concerning the problem of expressing measures according to proper scales and converting some scales into more suitable ones for pattern classification algorithms allows one to apply these algorithms more efficiently. This part of the theory of feature selection is well established and a variety of methods is defined [23].

3.3 Classification method analysis

The choice of a proper method is influenced strongly by two main determinants: a degree of irregularity of clusters (complexity of a particular problem), and a number of elements in a training set. If the degree is high, then very sophisticated methods have to be applied, and a big number of training set elements is required. For example, in the case of using parametric methods a lot of sample points representative for all the classes is needed in order to reconstruct distributions and *a priori* probabilities of the classes. Thus, a quantity of pattern samples restricts, somehow, the possibility of using sophisticated methods.
The *prototype classification* method based on the *nearest neighbour rule* [16] has been chosen for designing Recognize Anomaly process. Such a choice has been influenced by the following facts:

- a small number of the initial training set points (it increases as the experiment goes on),
- the necessity of an immediate response of the system (the method, as we will see, is computationally very efficient),
- the possibility of initial establishing of some class prototypes on the basis of a general knowledge of the DAQ system behaviour,
- the possibility of using the results produced with this method as a starting point for implementing more sophisticated methods (e.g. for constructing piecewise linear discriminants),
- the possibility of extending the method by the sequential approach.

The general idea of the nearest neighbour approach may be expressed shortly by defining discriminants functions in the following way:

$$g_i(\mathbf{x}) = -d(\mathbf{x}, \mathbf{y}_k^{(i)}), \tag{1}$$

where $\mathbf{y}_k^{(i)}$ is the nearest to $\mathbf{x}$ training set point belonging to the class ω^i, $d(\mathbf{x}, \mathbf{y})$ is the distance function. In a prototype classification we assume that we have an ideal representant (prototype) $\mathbf{m}_i$ for each class ω^i. We treat training set points belonging to this class as distortions of the representant. An unknown pattern is ascribed to the class, whose prototype is the nearest one. Thus, discriminant functions are defined as follows:

$$g_i(\mathbf{x}) = -d(\mathbf{x}, \mathbf{m}_i). \tag{2}$$

The advantage of the method is that it is sufficient to use linear discriminant functions for testing whether

$$d(\mathbf{x}, \mathbf{m}_i) < d(\mathbf{x}, \mathbf{m}_j). \tag{3}$$

If we use Euclidean metric, this is equivalent to

$$\sum_{n=1}^{N}(x_n - m_{in})^2 < \sum_{n=1}^{N}(x_n - m_{jn})^2, \tag{4}$$

where $\mathbf{m}_i = (m_{i1}, \ldots, m_{iN})$. It can be rewritten as

$$\sum_{n=1}^{N} x_n^2 - 2\sum_{n=1}^{N} x_n m_{in} + \sum_{n=1}^{N} m_{in}^2 < \sum_{n=1}^{N} x_n^2 - 2\sum_{n=1}^{N} x_n m_{jn} + \sum_{n=1}^{N} m_{jn}^2. \tag{5}$$

Thus, linear discriminant functions are of the following form

$$g_i(\mathbf{x}) = \sum_{n=1}^{N} x_n m_{in} - \frac{1}{2}\sum_{n=1}^{N} m_{in}^2, \; i = 1, \ldots, L. \tag{6}$$

As we have mentioned above, a sequential approach [21, 22] alllows one to divide a decision process into substages such that at each of them the final decision can be made in certain cases. This approach is especially useful if the data processing time is a crucial factor during recognizing. Let us assume that the features have been ordered according to their importance, and the accuracy of a classification on the basis of first k features can be evaluated. Then, in the two-class case the Wald's sequential probability ratio computed at the kth stage of a decision process (i.e. computed on the basis of k first features) is equal to

$$\lambda_k = \prod_{n=1}^{k} \frac{p(x_n/\omega^1)}{p(x_n/\omega^2)}. \tag{7}$$

To make a decision, we should earlier determine the following two thresholds

$$t_1 = \frac{1 - c_{21}}{c_{12}} \quad and \quad t_2 = \frac{c_{21}}{1 - c_{12}}, \tag{8}$$

where c_{ij} is the *cost* of deciding that $\mathbf{x}$ belongs to ω^i, whereas $\mathbf{x}$ belongs to ω^j.
Now, we classify $\mathbf{x}$ as belonging to ω^1, if $\lambda_k \geq t_1$, we claim it belongs to ω^2, if $\lambda_k \leq t_2$, and we require an additional $(n+1)$th feature, if $t_2 < \lambda_k < t_1$. The classification rule defined in such a way is optimal in a two-class case [21].

Let us determine the decision boundary for a two-class case, assuming that $x_1, \ldots, x_N$ are independent features, $p(x_n/\omega^i)$ are univariate gaussian density functions with a mean m_i and a variance σ_2. Then, let us evaluate $\log \lambda_k$, because of its better computational complexity property. Logaritmizing (7), we obtain

$$\log \lambda_k = \sum_{n=1}^{k} \log \frac{p(x_n/\omega^1)}{p(x_n/\omega^2)}, \tag{9}$$

so

$$\log \lambda_k = \sum_{n=1}^{k} \log \frac{(\sigma\sqrt{2\pi})^{-1} \exp[-(2\sigma^2)^{-1}(x_n - m_1)^2]}{(\sigma\sqrt{2\pi})^{-1} \exp[-(2\sigma^2)^{-1}(x_n - m_2)^2]}, \tag{10}$$

hence

$$\log \lambda_k = \frac{1}{\sigma^2} \sum_{n=1}^{k} x_n(m_1 - m_2) - \frac{1}{2} \sum_{n=1}^{k} (m_1^2 - m_2^2). \tag{11}$$

At the end of our considerations let us notice that the decision boundary equation (11) is analogical to that, which can be obtained from the formula (6) of linear discriminant functions for the prototype classification (by taking $g_1(\mathbf{x}) - g_2(\mathbf{x})$).

4 Experimental results and their use for designing the recognition module

As we have mentioned in Section 2, the process Summarize Run has been implemented firstly, because it is a tool for both the analysis of the problem from the point of view of the pattern recognition theory and building Expert System data bases. In this section we present some conclusions drawn from

the analysis of the DAQ system data throughput performance made for runs 2284 - 4117.

There is 23 global measurements of the online DAQ system performance available from the Eventbuilder [2], which constitute a measurement space (most of them are defined multiply for various DAQ system components). Initially, the subset consisting of nine of them was suggested as the sufficient one for detecting an unusual system behaviour [2]. This meant a significant reduction of the measurement space dimensionality. The following quantities were included in the reduced space:

- for GSLT (Global Second Level Trigger): a frequency, and a period between two consequtive positive decisions,
- for each of the seventeen detector components: a period between two consequtive data records, a time delay in relation to a GSLT decision, a size of event data, a load of a transputer internal buffer at the interface to EVB, a load of a TPM (triple-ported memory) at the interface to EVB,
- for each out of twelve transputers interfacing EVB with TLT (Third Level Trigger): a load of a transputer internal buffer, a load of a TPM.

The dimensionality of the space proposed was still very high. At first, buffer loads were considered as the crucial performance measurements [2], and the first version of the Summarize Run (1.0) module was implemented on the basis of this assumption. Let us discuss the results of the DAQ system performance analysis dividing our considerations into two parts: one concerning a misbehaviour at the detector components side, and another relating to the TLT/IBM link side.

4.1 The detector components subsystem

Filling up of the internal buffers of transputers at the TLT side is the first symptom of the detector components subsystem misbehaviour. It is caused by slow component(s), which do not send data on time. This means that incomplete events have to be kept on the TLT side until the missing data are sent. If the period between two consequtive positive GSLT decisions is very short in relation to the period beteween two consequtive data from the slow component(s), then at the TLT side transputer internal buffers are filled up very quickly. When they are full, EVB cannot send new data from the components subsystem. Therefore, filling up of the components buffers

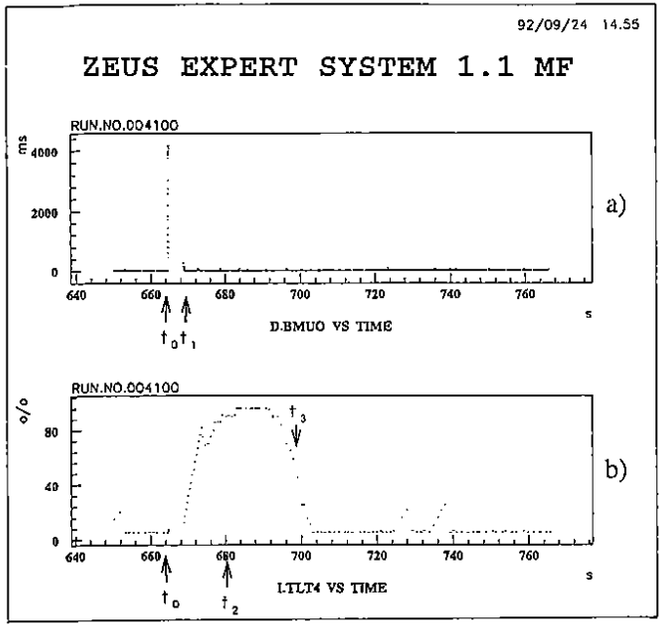

Figure 7: A component delay and its influence on buffers load

is the second symptom of a misbehaviour of this kind, which leads to the system blocking very quickly.

However, the load of buffers is a useful measurement for the DAQ system performance monitoring, it is an insufficient factor for predicting the system misbehaviour, because it reveals the symptoms, not the reasons. It means that at the moment of observing a phenomenon of buffers being filled blocking of the DAQ system has already started. The analysis of the abortive runs made with the help of the Summarize Run module has shown that in the case of a delay of one component even with low GSLT frequencies (20 - 30 Hz) there is only 2 - 3 seconds from the start of TLT side transputer buffers filling till the system is blocked.

"Fortunately", very often delays for all the slow components (processing of large data records) occur at the same time. It results in a longer time of buffers filling.

> *For example, during the run 4100 (the average GSLT frequency was ca 19.5 Hz) the trigger rate increased to about 31 Hz at 664 sec. of the run, which caused an excessive (about 4000 ms) delay of the barrel muon detector (see Fig. 7a, the value of the D.BMUO feature of a vector characterizing the system perfor-*

> *mance at t_0). Nevertheless, internal buffers of transputers at the TLT side were overloaded after 20 seconds, because a delay of calorimeter components occured at the same time (the increase of the I.TLT4 value within $[t_0, t_2]$ is shown in Fig. 7b). In spite of reducing the value of the D.BMUO feature at $t_1 = 668s$ (see Fig. 7a), the system came back to the normal state at $t_3 = 700s$ (see Fig. 7b).*

4.2 The TLT/IBM link

As far as the TLT/IBM link is concerned the only one measurement allowing us to "predict" the system anomalous behaviour is the load of TPM buffers (P.TLT), which is strongly corelated with the load of internal buffers at the TLT side (I.TLT) - see Fig. 8. In this case blocking all the links simultanously also occurs very often. However, contrary to the component side (which gives an input data), blocking all the output "sinks" accelerates I.TLT buffers filling.

> *The run 3375 is a typical example of a temporary blocking of this type. The average GSLT frequency of about 21 Hz was too high for the TLT/IBM link and after 1500 seconds of the run transputer internal buffers at the TLT side started being filled. It could be predicted that with such a frequency there are about 3.5 sec. until the buffers are full (at present there is a buffer space for ca 70 complete events at the TLT interface to EVB). The results from* Summarize Run *confirmed such a prediction.*

Thus, in this case there is very little time to react. In some cases, the occurrence of single "peaks" in P.TLT load plots (see Fig. 9a) or I.TLT load plots (see Fig. 9b) are warning signals for an approaching crash.

4.3 Discussion

The analysis of the DAQ system data throughput performance made for runs 2284-4117 with the help of a Summarize Run module has shown that blocking all the TLT/IBM links is the most frequent reason of the whole system low performance. Sometimes it has occurred altogether with an excessive delay of slow components. In such cases, it is usually very difficult to conclude what is the main (in the sense: primary) reason of the system low efficiency. Additionally, as we can observe TLT/IBM link misbehaviour

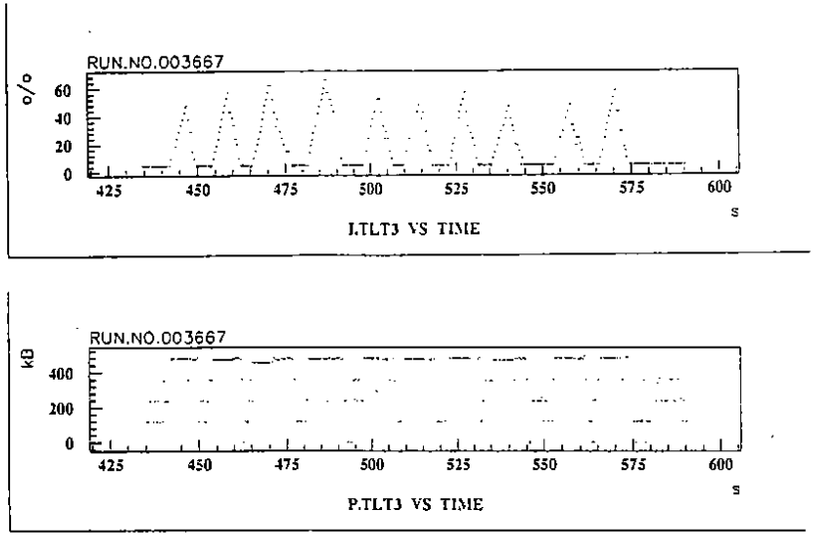

Figure 8: The corelation of TPM buffers load with internal buffers load at the TLT/IBM link

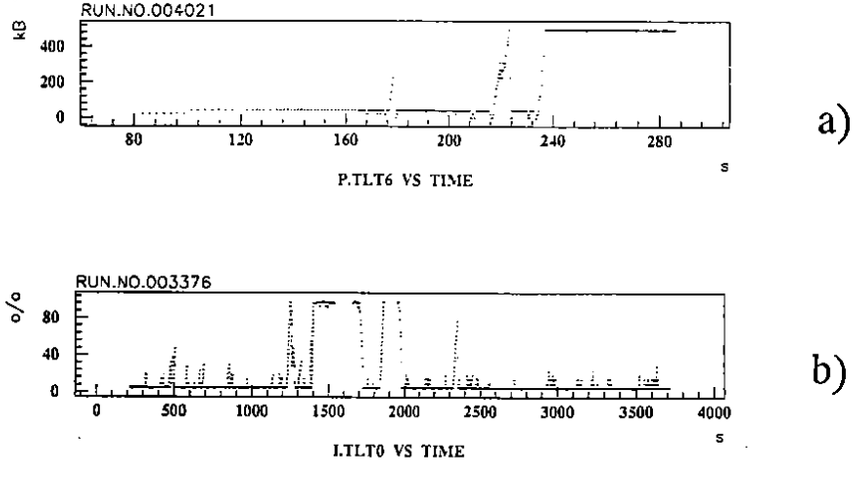

Figure 9: The preceding symptoms of TLT/IBM link blocking

only by symptoms (which gives very little time to react), recognizing the DAQ system misbehaviour should be focused on the load of TPM buffers at the TLT side. It is made by the Detect Anomaly process having the highest priority in the Expert System. The pattern recognition process is performed sequentially. At the first step, it is decided (scanning the features of the most importance, i.e. P.TPMs), whether there is a danger of TLT/IBM blocking. The P.TPM features are transformed to the new discrete ones, which are measurements of the danger of the system crash (see Subsection 3.2). If they exceed the threshold, the Expert System goes into the emergency mode (as described in Subsection 2.3). At the second step of the pattern recognition process, the delays of components responses to positive GSLT decisions are scanned. The analysis has shown that other measurements are strongly corelated with the ones mentioned above. Therefore, they cannot be directly used in a classification process. On the other hand, they can give some additional information, e.g. concerning the speed of approaching system blocking.

5 Concluding remarks

Due to the use of structural analysis and prototyping methodologies altogether with the analysis of the problem on the basis of the pattern recognition theory, there has been possible to select DAQ system performance measurements (i.e. to construct the feature space) indicating an unstable work of the system leading to its blocking. Implementing the module summarizing runs has allowed us to select and store "the histories" of the runs, which are interesting from the point of view of a designer of the Expert System knowledge data bases much before the first version of the whole system is ready. The assumption of a very simple prototype classification method has been a good starting point for further extensions and modifications (as the sequential one presented here), which is a common methodological practice in pattern recognition [23, 24].

The ZEXP formal model differs from models of conventional (no real-time) expert systems. However, some ideas used in ZEXP can be found in real-time expert systems. Doraiswami and Jiang [25] applied a real-time performance monitoring and preprocessing to transform obtained data to a form compatible with the knowledge base. The use of pattern recognition in expertizing is described in [26, 27]. Multi-level expertizing depending on time constraints has been introduced in [28]. For simple and quick detecting

the forward-chaining is applied in Doraiswami and Jiang expert system [25], whereas the backward-chaining procedure performs a complex analysis, if additional time is available.

As we have mentioned in Introduction, designing a version of ZEXP on the basis of EVB monitoring data limits its expertizing abilities. We have assumed such a solution, because EVB monitoring is well understood. On the other hand, there are additional information sources, e.g. GFLT/TLT monitoring, slow control, component monitoring, which could be used to enhance ZEXP expertizing abilities. The first two are currently under investigation. The real-time expert systems are constructed as a diagnostic, analytic, or even automatic control tools. The prototype implementation of ZEXP is a diagnostic one with some analytic features, which are being developed. At the moment there is no possibility for implementing the automatic control tool, because of changing , instable DAQ conditions. A detailed discussion of prospects of an Expert System for ZEUS is included in [29].

A Description of main Summarize Run modules

EXP_prunsum2.csh

Language: UNIX shell.
Description: The main module calling procedures: creating N-tuple for the Physics Analysis Workstation (PAW) package, and preparing the command file for PAW. It calls PAW and sends the produced output files for printing.

pprepPAWcom2.csh

Language: UNIX shell.
Description: The shell script preparing the command file for PAW.

mkntp

Language: C
Description: The module cretaes Ntuple containing both memory usage and time characteristics monitored during a run. Ntuple is created with HBOOK package.

References

[1] C. Youngman, *Necessary improvements to the OnlineSystem*, ZEUS-Note, *in preparation.*

[2] U. Behrens, L. Hagge, T. Schlichting, W.O. Vogel, *Status of the ZEUS Eventbuilder*, ZEUS-Note 92-054, DESY, June 1992.

[3] T. DeMarco, Structured Analysis and System Specification, Yourdon Press, New York, 1978.

[4] E. Yourdon, Modern Structured Analysis, Yourdon Press/Prentice-Hall, New York, 1989.

[5] D.J. Hatley, I.A. Pirbhai, Strategies for Real-Time System Specification, Dorset, New York, 1988.

[6] J. Magee, J. Kramer, M. Sloman, *Constructing distributed systems in Conic*, IEEE Trans. Software Engineering **15** (1985), 663-675.

[7] S. Goering, S. Kaplan, *Visual concurrent object-based programming in GARP*, Lecture Notes in Computer Science **366** (1989), 165-180.

[8] M. Flasiński, L. Kotulski, *On the use of graph grammars for the control of a distributed software allocation*, The Computer Journal **35** (1992), A167-175.

[9] M.M. Tanik, R.T. Yeh, *Rapid prototyping*, IEEE Trans. Computer **22** (1989), no 5.

[10] J. Connell, L. Schafer, Rapid Prototyping, Yourdon Press/Prentice-Hall, New York, 1989.

[11] U. Behrens, M. Flasiński, L. Hagge, J. Jurek, K. Ohrenberg, G. Sawicki, *DAQ-to-ZEUS Expert System interfacing*, ZEUS-Note, *in preparation.*

[12] E. Rich, K. Knight, Artificial Intelligence, Mc Graw-Hill, New York, 1991.

[13] E. Feigenbaum, P. Mc Corduck, The Fifth Generation, Reading, Mass., Addison-Wesley, 1983.

[14] M. Flavin, Fundamental Concepts of Information Modeling, Yourdon press/Prentice Hall, New York, 1981.

[15] G. Sebestyen, J. Edie, *An algorithm for nonparametric pattern recognition*, IEEE Trans. Electronic Computers **15** (1966), 908-915.

[16] T.M. Cover, P.E. Hart, *Nearest neighbor pattern classification*, IEEE Trans. Information Theory **13** (1967), 21-27.

[17] J. Owen, D.B. Brick, E. Henrichon, *A nonparametric approach to pattern recognition*, Pattern Recognition **2** (1970), 227-234.

[18] M. Flasiński, G. Lewicki, *The convergent method of constructing polynomial discriminant functions for pattern recognition*, Pattern Recognition **24** (1991), 1009-1015.

[19] Y.T. Chien, K.S. Fu, *On Bayesian learning and stochastic approximation*, IEEE Trans. Systems Science Cybernetics **3** (1967), 28-38.

[20] H. Robbins, *The empirical Bayes approach to statistical decision problems*, Ann. Math. Statist. **35** (1964), 1-20.

[21] A. Wald, Sequential Analysis, Wiley, New York, 1947.

[22] K.S. Fu, *A sequential decision model for optimum recognition*, in Biological Prototypes and Synthetic Systems, vol. I, Plenum Press, New York, 1962.

[23] M.R. Anderberg, Cluster Analysis for Applications, Academic Press, 1973.

[24] M. Flasiński, *Distorted pattern analysis with the help of node label controlled graph languages*, Pattern Recognition **23** (1990), 765-774.

[25] R. Doraiswami, J. Jiang, *Performance monitoring in expert control systems*, Automatica **25** (1989), 799-811.

[26] A. Carmon, *Intelligent knowledge-based system for adaptive PID controller tuning*, Journal A (Benelux Q. J. Aut. Control) **27** (1986), 133-138.

[27] B. Porter, A.H. Jones, C.B. McKeown, *Real-time expert tuners for PI Controllers*, IEE Proc. **134** Pt. D, 260-263.

[28] M.L. Wright, M.W. Green, G. Fiegl, P.F. Cross, *An expert system for real-time control*, IEEE Software (March 1986), 16-24.

[29] U. Behrens, M. Flasiński, L. Hagge, *Prospects of an Expert System for ZEUS*, ZEUS-Note, *in preparation.*