APPLICATION OF EXPERT SYSTEMS IN HIGH ENERGY PHYSICS: THE ALEPH AND ZEUS CASE

P. Mato
CERN, Geneva, Switzerland

Abstract
Two cases in which Expert System techniques have been used for the operation of large high energy physics experiments are presented. In both cases, the driving force has been to achieve better performance by increasing the efficiency and reliability of the online system. The scope and implementation of both systems will be described and the differences of the two approaches will be highlighted. Finally, a summary of the lessons learned by the deployment and use of these Artificial Intelligence methods in running a complex experiment will be given.

1. INTRODUCTION

The objective of this lecture is to illustrate, using two examples, the way that Artificial Intelligence (AI) techniques, in particular Expert Systems, have been used to assist shift crews in the operation of currently running High Energy Physics (HEP) experiments. These two examples are ALEPH and ZEUS. ALEPH is one of the four experiments at the LEP collider at CERN, and ZEUS is an experiment located at HERA, the electron-proton colliding facility of DESY. The deployment of these Artificial Intelligence (AI) techniques required solving practical problems. The experience gained can perhaps be used during the design of the new generation of DAQ systems for experiments that are under preparation.

The lecture comprises four parts. The first part is a general review of what motivated the two experiments to start Expert System projects. The second and third parts are the reviews in some detail of the ALEPH and ZEUS expert systems: DEXPERT and ZEX. For each one, the analysis of the particular requirements, the design chosen and the way it has been implemented is reviewed. Finally, in the last part, we look to the future and try to see what can lessons we have learned that may be applied to the new generation of experiments.

2. MOTIVATIONS FOR EXPERT SYSTEMS

Both ALEPH and ZEUS collaborations decided at a given moment to launch an expert system project. The goals established by both collaborations were to increase the efficiency and reliability of the operation of the experiment, and to allow a reduction of the manpower needs and level of expertise of the shift crew. The efficiency can be increased if errors or anomalous situations are automatically handled and recovered. This is especially true if the recovery is done in less time than it would take an average trained shifter to perform the recovery manually. Concerning the reduction of manpower in the particular case of ALEPH, it was decided by the collaboration to run the experiment with two people in the control room without a DAQ expert on shift.

Another motivation was to study the applicability of Artificial Intelligence (AI) techniques, in particular of expert systems, in HEP experiments. It was also important in the case of ZEUS to see how computer science theories like *pattern recognition* and *graph grammars* could be applied to solve practical problems in running experiments.

2.1 Handling Complexity

The ALEPH and ZEUS experiments are representative examples of large HEP experiments of the early 90's. They consist of a large detector with about ten sub-detectors, hundreds of physicists, a large DAQ system with hundreds of crates and processors, etc. The diversity of hardware and software components is also something that is remarkable. For example, the number of different programs that are running concurrently during data-taking to perform the various functions is of the order of a hundred. The interactions between the various sub-systems like DAQ, trigger and timing, detector control, data monitoring, safety, etc. are also complex. An anomalous behaviour of one element in one sub-system may affect in a non-trivial manner other sub-systems.

These experiments have been in operation for unprecedented periods. During their lifetime, the DAQ system and the environment have been in continuous change. Addition of new sub-detectors, changes in hardware or software, upgrades of the operating system, etc. It is clear that nobody can be an expert of everything. Knowledge is distributed and unfortunately knowledge is evaporating each time people in charge of parts of the experiment are replaced. In addition, physicists who are not in general experts on the trigger, DAQ, detector, etc, are operating these experiments.

2.2 Improving Performance

The first thing we need to do before we try to improve the performance is to measure it. The efficiency of the Trigger and DAQ system is measured by the ratio of interesting physics events collected and stored to the number of events produced by the accelerator. This efficiency is expressed as the product of various efficiency factors weighted by the luminosity of the machine. The DAQ efficiency factor is the fraction of the time the DAQ system is operational. To improve performance we need to minimise the time the Trigger and DAQ system is not operational. Expert systems may play a role in that since they can provide diagnosis and recovery of problems faster than a human operator can.

3. DEXPERT

3.1 Scope and Requirements

The operator controlling the ALEPH DAQ system is in charge of performing the start and stop sequences and also of handling the error conditions that may occur during data-taking. The origins of these errors are violations of the protocols that dictate the behaviour of the different components of the system. These protocols maybe violated because either hardware or software malfunctions. Upon detection of an error, any task in the system can force the *run controller* to go into an error state, automatically disabling the trigger. At this moment the operator has to diagnose the error and apply the corrective actions to resume data-taking. DEXPERT (DAQ Expert System) was developed to assist the operator with the recovery from read-out errors.

The main requirement for DEXPERT is that it should emulate as closely as possible the behaviour of a human expert operating the ALEPH DAQ system. As the human operator does, DEXPERT should react automatically to read-out errors, apply its knowledge about this particular problem domain, receive error messages, access databases, perform corrective actions using the same control programs as the operator, and finally, restore the running conditions (see Figure 1). DEXPERT gets interrupted when an alarm or error message is received from the general ALEPH *Error Logger* which is in charge of collecting and logging all the errors, which are produced by any part of the system. It is not the role of DEXPERT to monitor the system and detect the anomalies. The error detection is done at the source. It should also be possible for the operator to enable/disable DEXPERT like an autopilot and intervene if it goes out of control.

The constraints for DEXPERT included that it should be integrated within the existing ALEPH DAQ system. This implies using the ALEPH standard packages for communication, user interface, database access, etc.

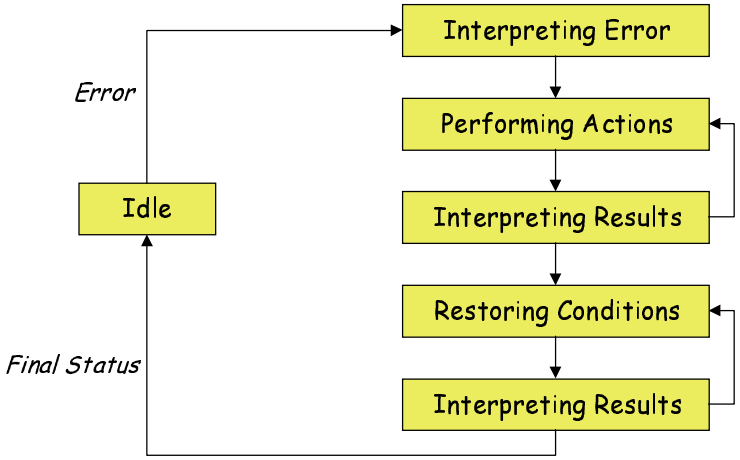

Figure 1 DEXPERT general behaviour

3.2 Design and Implementation

DEXPERT was designed with two well-differentiated parts: the thinking part and the interface part. An overview of the design is shown in Figure 2.

- The thinking part, the *Brain*, is where the problem is analysed and decisions are taken. This part is implemented using an expert system tool.
- The interface part is composed of a number of independent objects called *Tentacles*. Each *tentacle* is specialised to interact with a specific component of the system (i.e. one knows how to talk to the *run controller*, to the trigger controller, to the operator, etc.). In addition a more general object called the *Cerebellum* that serves as a bridge between the *Tentacles* and the *Brain* is also needed.

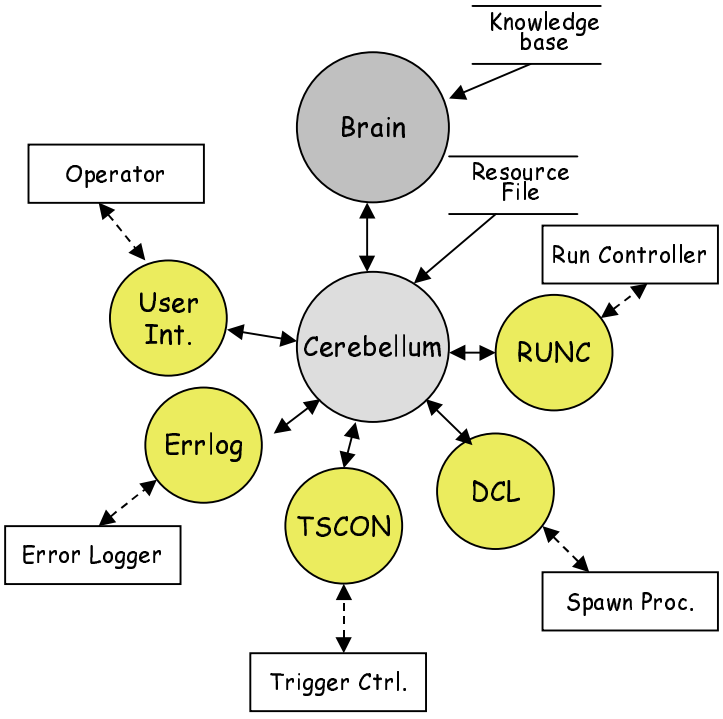

Figure 2 DEXPERT overall architecture

DEXPERT can reason about three basic data types: *Alarms*, *Actions* and *Action replies* (see Figure 3).

- The *Alarms* are generated by the *Tentacles* after reception of external or internal stimuli. They are collected by the *Cerebellum* which queues them to the Brain triggering the start of reasoning.
- The *Actions* are the results of the reasoning process of the *Brain*. They are queued to the *Cerebellum* who dispatches them to the *Tentacle* that is in charge of ensuring that that particular action is executed.
- The *Action replies* of the scheduled *Actions* are collected by the corresponding *Tentacle* and are sent back to the *Brain* through the *Cerebellum* to allow reasoning to resume.

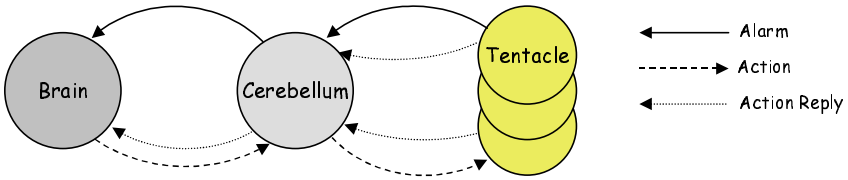

Figure 3 Basic data types being exchanged between DEXPERT components

Each of the main components of DEXPERT is modelled as a finite state machine (FSM). Figure 4 shows the FSMs for the *Cerebellum* and *Tentacle*. Each component can proceed in an asynchronous way as if they had a life of their own, allowing DEXPERT to be able to execute several actions at the same time.

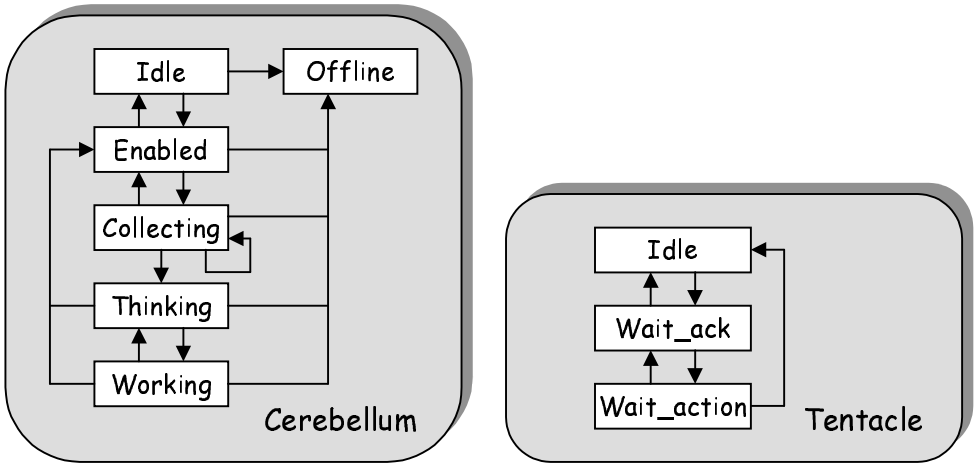

Figure 4 Finite State Machine for the *Cerebellum* and the *Tentacles*

DEXPERT is implemented as a VMS process running in one of the ALEPH on-line computers. The *Cerebellum* and *Tentacle* components and the basic data types are implemented as C++ classes. The *Brain* is implemented as a C++ class wrapper to a rule-base production system written in the OPS5 [4] language. In the following sections we will go into more in depth the way in which the *Brain* has been implemented. Starting with the criteria for selecting such a tool, what the tool consists of and the kind of expertise we have been able to program in the *Brain*.

3.2.1 DEXPERT Brain

One of the main practical requirements for selecting what expert system tool to choose in order to implement the *Brain* was that the tool must be callable from outside (start thinking when errors and

alarms have been collected) and also have the possibility to call external functions which are needed when accessing databases, performing actions, etc. The tool should support *forward chaining* since in this application we do not need to know the exact origin of the symptoms in order to apply the recovery actions. From facts we deduce new facts and apply recovery actions.

The OPS5 programming language was selected since it fulfils the requirements and also it is well integrated in to the VMS operating system. OPS5 is a powerful pattern-matching language developed at Carnegie Mellon in the late 70's. It has been used to develop large industrial knowledge-base systems. An example of a production rule that is used in DEXPERT is shown in Figure 5.

```
(P TRIGGER_ERROR::TMO_Wait_No_Busy
  { <MODULE> ( MODULE ^NAME   TRIGGERERROR
                      ^LEVEL  <L>
                      ^STATE  GO )
  { <ALARM>  ( ALARM  ^ERRORNAME TRIGGERERROR
                      ^P1        |TMO_Wait_No_Busy|
                      ^P2        <P2>
                      ^P3      { <P3> <> MANY } ) }
-->
  ( CALL BRAIN_GET_INFO FIOD <P2> <P3> )
  ( CALL BRAIN_ERRMSG |Trigger_Error::TMO_Wait_No_Busy>> Doing a FIOD| )
  ( MAKE TRIGGER_ERROR::HANDLE_BUSY_TMO )
)
```

Figure 5 Example of a production rule used in DEXPERT

The expertise of DEXPERT can be classified in to 3 types: Heuristic knowledge, decision trees and recovery sequences. Most of the knowledge is of the heuristic type. Very often it does not need to know the real cause of the problem (full diagnosis) to be able to execute the proper recovery actions. The rules of the DEXPERT production system are chained to produce decision trees to diagnose sufficiently the problem up to the level of being able to select the proper recovery. Usually the recovery of a problem requires a sequence of actions to several parts of the system. These sequences are also part of the knowledge base.

There are about 250 rules in DEXPERT. Due to some limitations in OPS5 that considers any rule at the same level as any other, there was the need to put some effort in to managing these rules in a more modular way. An example of that is the set of rules needed for sequencing the actions for a given recovery. This set of rules could be called from various decision trees and the executed sequences should not get mixed.

3.3 Operating DEXPERT: Successes and Failures

The development of DEXPERT was done fast and quickly put in production. It is able to handle about 90-95% of the possible errors during data-taking. The difficult problems for which there is not a well established recovery are simply not handled and given up passing the control to the human operator.

DEXPERT was extremely useful during the first years of running the ALEPH experiment, since the number the errors was higher than now (presently there are less "bugs" in the software and better hardware). It clearly fulfilled its original goals (increased efficiency and allowed running without a DAQ expert on shift).

One of the problems that ALEPH encounters now is that the average shift crew is less knowledgeable of the system and relies heavily on the expert system solving the problem. Also some

sort of failure is that the usage of these expert system techniques has not been extended to other parts of the system like the *slow controls*. However, the big problem is the difficulty to maintain the knowledge base. Only experts knowing the OPS5 language can do this. The language is complicated and intrinsically difficult to debug. Ideally, an interface could have been built to enter the "expert knowledge" in an easy way, i.e. using a graphical user interface, and then producing automatically OPS5 code which then could be compiled into DEXPERT. This kind of interface program was never realised.

4. ZEX

4.1 Scope and Requirements

The ZEUS collaboration decided in 1992 to construct an expert system to support the operating of the experiment. ZEX stands for ZEUS Expert System. The goals were very similar to those of ALEPH. Firstly to increase the efficiency and reliability of the experiment and secondly to store the knowledge of various real experts of the experiment and to make it available to everyone. The project was divided into several stages. The first stage was the development of the ZEX prototype initiated at the end of 1992 and put into operation in 1993. The ZEX prototype was used for diagnosing pre-selected aspects of the experiment data transmission. Based on this experience, the development of the extended system (ZEX) covering all key areas of the experiment was started in 1994 and put into production in 1996.

ZEX covers more areas of the experiment than DEXPERT. In particular the "slow controls" are covered by ZEX and not by DEXPERT. Concerning the phases in the processing of errors: (a) monitoring the DAQ system for symptoms of anomalous behaviour, (b) finding the origin of this anomalous behaviour (diagnosis) and (c) recovering the DAQ system from errors, ZEX focusses primarily on (a) and (b) while DEXPERT focuses on (b) and (c).

The system interfaces, which embed ZEX into the online DAQ system, are shown in Figure 6. Input to the expert system comes from the various sub-systems, output is to the DAQ system. The knowledge of several experts is required to prepare and tune ZEX. Experts from the different components provide detailed knowledge about the monitored quantities. A knowledge engineer prepares the routines for the general purpose analysis, and the DAQ coordinator collects knowledge about case specific treatments.

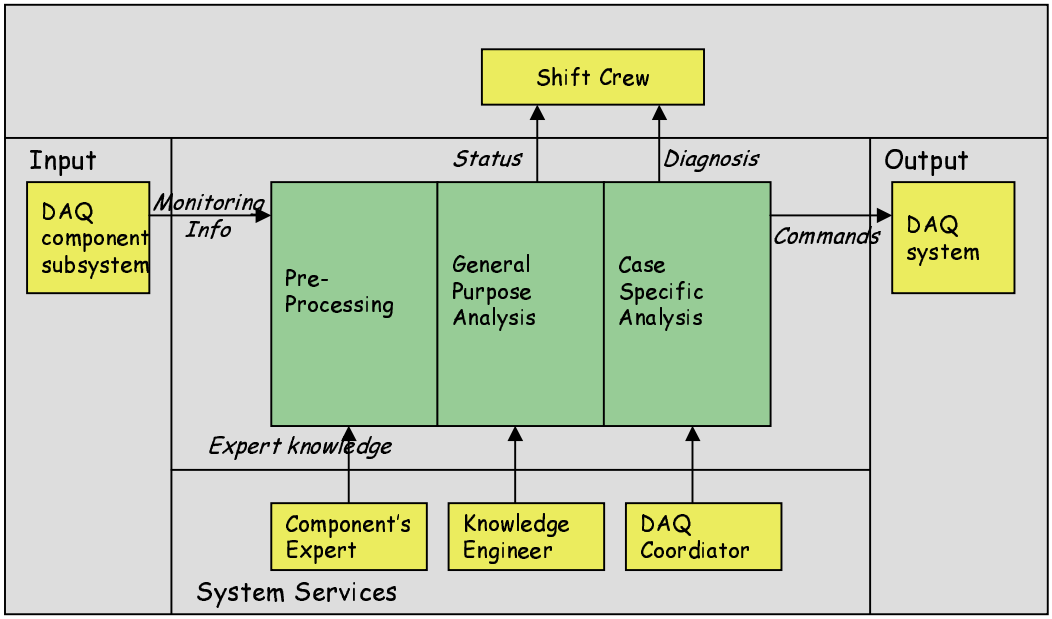

Figure 6 ZEX system environment

4.2 Design and Implementation

ZEX has been designed using OO methodology. The design model allows one to define the knowledge base of any expert system in accordance with basic OO principles of abstraction, encapsulation, modularity and hierarchy. This methodology has been applied to both parts of the knowledge base: *entity-level knowledge* (solution space) containing such objects as input data, partial solutions, final solutions and control data, and *problem-solving knowledge* that is the set of interpretative procedures used for reasoning over data in the solution space.

An approach called *Blackboard* has been used to design ZEX. In this approach the system is partitioned into (see Figure 7):

- A global hierarchical data structure of a *solution space* called *Blackboard*.
- Independent hierarchically organised *Knowledge-Sources* (KS) containing *problem-solving knowledge*, which run under the *controller*.

The *entity-level knowledge* of ZEX is stored in the Blackboard according to a hierarchical structure (system → subsystems → components)

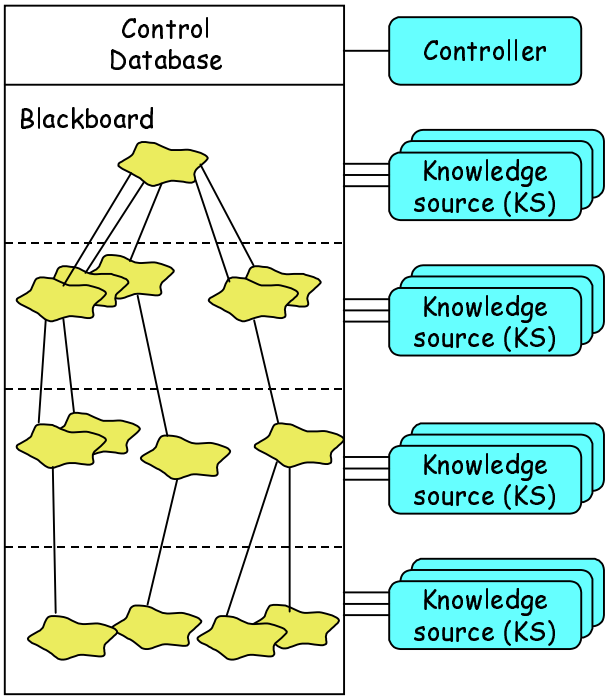

Figure 7 Blackboard architecture schema for ZEX

The *problem-solving knowledge* (interpretative procedures) in ZEX is modularised and encapsulated in Knowledge-Sources (KS). Each KS can be either Rule-based knowledge (production system) or Patter Recogniser-based knowledge as shown in Figure 8.

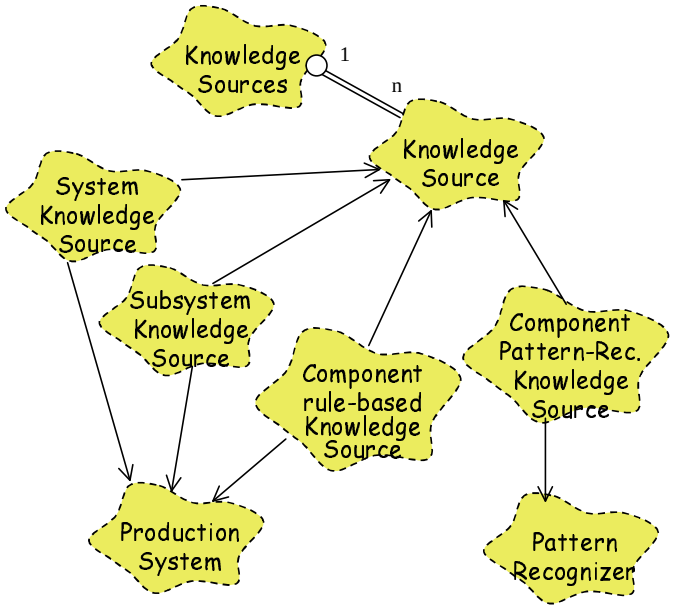

Figure 8 Knowledge Sources class diagram for ZEX

4.2.1 ZEX Expertise

ZEX is a hybrid expert system. Two different techniques are used together to express the problem-solving knowledge in ZEX: *Syntactic pattern recognisers* and *rule-base production systems*. The first is mainly used to detect the symptoms of anomalous behaviour and the later to perform the diagnosis of the problem.

A *production system* consists of a working memory containing assertions (statements that certain facts are true) that correspond to input data and intermediate results, a set of rules of the form "if <condition> then <actions>", and a rule interpreter or inference machine that evaluates the rules if any assertion changes in the condition part. Conflict resolution strategies are needed to select the order in which rules are executed in case of conflict. In case of real-time applications most assertions are treated as functions of time, i.e. it is necessary to consider the present values of process data as well as the past ones.

The production system of ZEX has been implemented using the *RTworks* shell from Talarian. RTworks is a family of software products for building client/server applications that intelligently manage time-critical data.

A rule-based system is not adequate for pre-processing the monitoring information (feature extraction) and the general analysis of features to detect anomalies. For this kind of processing, ZEX uses a *Syntactic Pattern Recogniser*. This consists of a signal processor filtering monitored data, a cluster classifier that classifies observed phenomena into predefined classes identified by a symbol and finally an automaton that reads the string of these symbols and recognises the state of the phenomenon in time series.

4.3 Operating ZEX

ZEX consists of about 100 rule-based *Knowledge-Sources*, containing more than 1500 rules implemented with the commercial shell RTworks and several syntactic pattern recognition-based *Knowledge-sources*. The last version was commissioned in 1995 and has been in production since 1996.

4.3.1 The truth

Neither DEXPERT nor ZEX are as good stories as they may seem to be. For instance ZEX will be discontinued this year. The ZEUS collaboration has decided to switch off ZEX in 1998 because the cost of the maintenance (licences and manpower) is higher than the practical benefits (the automatic recovery was not fully implemented and the operator had to do the recovery by hand). Concerning DEXPERT, it is in operation and is still giving satisfaction but other "clever" elements (handling power supply trips) not based on AI techniques have been introduced into ALEPH. In fact these are written using a procedural language, i.e. FORTRAN. This type of expertise in the area of "slow control" could have been added into DEXPERT but was not done.

The reasons for the disappointing end to these stories are somehow related to the sociology of big collaborations and are not associated to the AI techniques themselves. The HEP collaborations are big, and not everybody is convinced of the advantages that this kind of technology can bring to the success of the experiment. This is similar in some ways to the questioning of the advantages of using an OO approach for software development coming from members of HEP collaborations. The other reason is that experiments run for many years and people are not permanently attached to their developments. Information somehow evaporates and so it is not unusual that some things are re-invented during the lifetime of the experiment. Finally, there is a huge inertia to introduce new computing techniques. HEP experimentalists are reluctant to introduce new computing techniques in general unless they see clear benefits.

5. LOOKING TO THE FUTURE

We must learn from the successes and failures of these pioneering experiences on using expert system techniques to assist operators running large HEP experiments. We need to apply the lessons learned during the design and implementation of the new generation of experiments, in particular we need to focus on the LHC experiments, since the future of CERN is LHC.

The *Experiment Control System*[1] for the general purpose experiments, ATLAS and CMS are at least one order of magnitude larger than those of the LEP experiments. That is in number of sub-systems, number of parameters, diversity of equipment, etc. Operating such experiments will certainly require the aid of expert systems. We face two especially challenging problems: interfacing the Expert System with the *Experiment Control System* and the knowledge acquisition and long-term maintenance.

5.1 Interfacing the Expert System

It is essential to have an architecture from the beginning that foresees the intelligent assistance of the operators in charge of running the system. The Expert System should be an integral part of the system, i.e. one of many components that constitutes it. Integrated does not necessarily mean monolithic. An analogy could be to say that a *spelling checker* is an integral part of any word processor, however you can run a word processor without having a *spelling checker*.

The Expert System needs to interact with the central Error/Alarm handler of the experiment. It also needs to have access to all current and past status and monitor information in a coherent way. It is impractical if for each part of the system and sub-system the Expert System requires a different type of interface. And finally, it needs to have access to the configuration database for all the system. The proposed architecture for the *Experiment Control Systems* for the LHC experiments is shown in Figure 9. It foresees to have an Expert System component at the same level as other components of the system like the *Alarm Handler, Data archiver,* etc.

---

[1] The term Experiment Control System refers to a control system that controls and monitors everything in the experiment. It includes the detector control or slow control, run control, physics data monitoring control, etc.

It is essential for interfacing the Expert System that all the functions and components of the system provide an Application Programming Interface (API). For example, it is no good if the only way to perform a given function is by means of using the mouse. If that was the case the Expert System would never be able to emulate an operator.

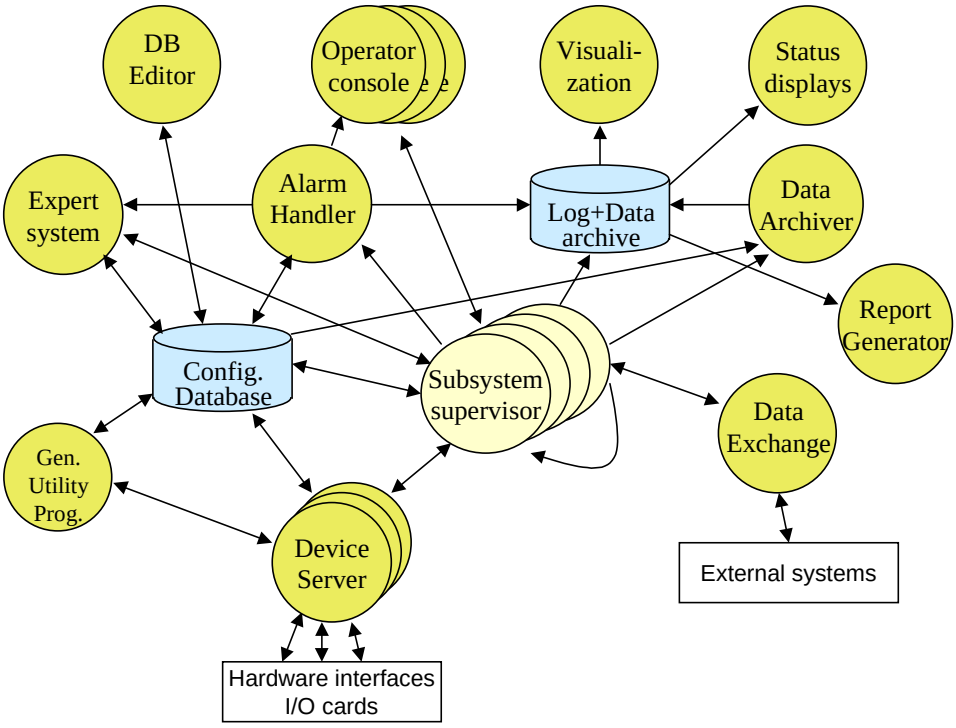

Figure 9 Proposed Experiment Control System architecture for the LHC experiments

5.2 Knowledge Acquisition and Long-term Maintenance

Knowledge acquisition is the really challenging problem. It is inherently difficult to introduce knowledge into an Expert System. On top of that, experiments are not static; they are in continuous evolution, therefore the knowledge that is useful for the way you operate the experiment at a given moment may not work when the experiment is changed or upgraded. Only “real experts” can introduce their expertise, but not all the experts are ready to dive in with a complicated language (the case of DEXPERT) or interface.

Self-learning techniques are very attractive. The Expert System could observe the DAQ and Control system and at the same time it could spy what actions the operator is performing to solve problems and deduce the rules (learning). This technique has not been proven in our environment and it may happen that the effort needed to implement such a schema may not justify the potential benefits.

Sophisticated login of actions and errors is essential. Knowing and classifying the different problems which occur while running the experiment offers a good tool for improving the system. It is fundamental to know which problems cause the most inefficiency, and thus need to be worked on if one wants to improve the efficiency.

The life-time of the LHC experiments will be more than 10 years. The Expert System should be able to cope with changes and upgrades of the system. People other than the developers will run the experiment. It is clear that there is no silver-bullet solution for this kind of problem, however several precautions can be taken from the beginning. For example, designing the overall system with an Expert System as an integral part (later add-ons are very often problematic), providing tools that

make the introduction and changes of knowledge simple, and finally, allocating manpower for the evolution of the knowledge base.

6. CONCLUDING REMARKS

The ALEPH and ZEUS experiences are positive in demonstrating the benefits of applying AI techniques, in particular Expert Systems, for the operation of HEP experiments. The scope and the aims were a bit different for both systems. And if one look at them with some perspective, the conclusion could be that aiming at something simpler that produces clear benefits has better chances of long-term success than something sophisticated.

For the LHC experiments, Expert Systems to assist the operator running the experiment are a must. Therefore, the DAQ and Control systems need to be designed with intelligent and automated assistance in mind from the beginning.

ACKNOWLEDGEMENTS

Many people from the ALEPH and ZEUS collaborations have participated on the development and maintenance of both DAQ systems and Expert Systems. The material of this course is essentially based on their work.

BIBLIOGRAPHY

[1] W. von Rüden, The ALEPH data acquisition system, IEEE Trans. Nucl. Sci. 36 (1989) pp 1444-1448

[2] A. Belk et al., DAQ software architecture for ALEPH, a large HEP experiment, IEEE Trans. Nucl. Sci. 36 (1989) pp 1534-1539

[3] C. Youngman, The ZEUS data acquisition system, in Proc. CHEP'92, CERN 92-07, pp. 145-150

[4] L. Bownston et al., Programming Expert Systems in OPS5. An Introduction to Rule-Based Programming, Addison-Wesley, MA, 1985

[5] P. Mato, DEXPERT: An Expert System for Read-out Error Recovery in the ALEPH Data Acquisition System, Proc. New Computing Techniques in Physics Research II, World Scientific, 1992

[6] P. Mato, T. Wildish, Applying Object-Oriented, Real Time, and Expert System techniques to an automatic read-out error recovery in the ALEPH data acquisition system, Proc. Computing in High Energy Physics'92, CERN 92-07, 1992

[7] Talarian Corporation, Mountain View, USA, RTworks v 3.0 User Manual, June 1994

[8] R. Englemore, T. Morgan (editors), Blackboard Systems, Addison-Wesley, MA, 1988

[9] U. Behrens et al., ZEXP - Expert System for ZEUS Problem Analysis, Theoretical Background and First Results, DESY 92-141, October 1992

[10] U. Behrens et al., ZEX – An Expert System for ZEUS, IEEE Trans. Nucl. Sci. 41 (1994) pp 152-156

[11] M. Flasinski, Further Development of the ZEUS Expert System: Computer Science Foundations and Design, DESY 94-048, March 1994

[12] U. Behrens et al., Recent Developments of the ZEUS Expert System ZEX, IEEE Trans. Nucl. Sci. 43 (1996) pp 65-68