1 Introduction

The data acquisition system of ALEPH (one of the four experiments at the LEP collider) consists of a large number of processors (~80) distributed in approximately 140 Fastbus crates [1]. In each processor several programs running in a multi-tasking environment cooperate during the data-taking activity [2].

The operator controlling the data acquisition 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 which dictates the behavior of the different components of the system. In fact, the protocols are violated because of hardware and software misbehaviors. Upon detection of an error, any task in the system can force the run controller to go to 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.

The ALEPH collaboration decided to operate the experiment starting from 1991 with a smaller shift crew (i.e. 2-3 man). In particular it was decided to run with no data acquisition expert on shift since the number of these experts is gradually decreasing with time. This together with the fact that few different types of errors account for most of the errors which occur during data taking suggests that an automatic error recovery program would be desirable and possibly feasible. Therefore a rule-based expert system program (DEXPERT) was developed to assist the data acquisition operator in his job.

The main goal for DEXPERT is that it should emulate as closely as possible the behavior of a human expert operating the Aleph data acquisition system. Since the human behavior is not known for all the possible situations the program must be very flexible. As the human operator does, DEXPERT should react automatically to read-out errors, apply its knowledge about this problem domain, receive error messages, access data bases, perform corrective actions using the same control programs as the operator, and finally, restore the running conditions (Figure 1). It should also be possible for the operator to enable/disable...
DEXPERT like an auto-pilot. The operator can always intervene if DEXPERT goes out of control.

The only constraint we had is that DEXPERT should be integrated within the existing Aleph data acquisition system. This implies to use the Aleph standard packages (i.e. communication package, user interface, databases, etc.).

![Diagram](image)

Figure 1 General behavior of DEXPERT

2 Design and Implementation

2.1 Design

DEXPERT has been designed with two well differentiated parts: the thinking part and the interface part. An overview of the design is show in Figure 2

- The thinking part, the brain, is where the problem (read-out error) is analyzed and the decisions are taken. This part is implemented using an expert system tool. This tool should ease the programming of the knowledge.

- The interface part is composed of a number of independent objects called tentacles. Each tentacle is specialized on the interaction with a well defined component of the system (i.e. the one that knows how to talk to the run controller, to the trigger controller, to the operator, etc.). Moreover a more general object called cerebellum which serves as a bridge between the tentacles and the brain is needed.

We expect that DEXPERT will be in a continuous changing phase. This is because we can not predict a priori all possible errors that may occur and what is even more difficult we cannot predict all possible recovery procedures. For this reason we need to design DEXPERT applying any good technique that will help during the maintenance phase. This is the case of O-O. Two aspects of O-O programming we found fundamental for designing DEXPERT:

- Information hiding helps to keep objects with very clear interfaces. For example, the brain object only needs to know about the cerebellum and its knowledge base.
- Although there are many *tentacles*, each one interacting with a different external partner, we can distinguish basic common behaviors among them. For example, all of them have the need to receive input from the outside world and they need to keep track of the state of its partner. By using *inheritance*, we can isolate these common behaviors creating some basic classes.

![Figure 2 DEXPERT overall design](image)

In order to reproduce the desired behavior, the objects we have defined need to interact by interchanging data of one of the three main data types.

- **Alarms** or error conditions. They are generated by the *tentacles* after reception of external or internal stimuli. The *cerebellum* acts as a concentrator of *alarms* which are then queued to the *brain*. The *alarms* trigger the *brain* inference machine.

- **Actions**. They are generated by the *brain* and are queued to the cerebellum which dispatches them to the appropriate *tentacle*. The *tentacles* are in charge of making the actions to be executed.

- Action replies. The replies for the scheduled actions are collected by the *tentacle* and are sent back to the *brain* through the *cerebellum* to allow the reasoning process to resume.

2.2. Implementation

The OPS5 programming language [3] was chosen to implement the *brain*. The OPS5 language is used in the field of artificial intelligence, mainly to develop expert systems. It uses the production rules formalism. The rules have the form of *condition-action* (if-then). It has been no problem encapsulating a OPS5 program into an object. The interface part of DEXPERT has been implemented using the C++ programming language [4].

The complete DEXPERT is *event-driven*. For each external/internal stimulus (i.e. message, I/O, operator input, timer, etc.) an asynchronous event is inserted into an *event*
queue which is then directly dispatched, using a set of input specific objects called sensors, to the object that has requested to receive it.

![Diagram of DEXPERT](image)

Figure 3 Dispatching asynchronous input to the object interested on it.

Each main object in DEXPERT (i.e. tentacles, brain, cerebellum, etc.) is modeled as a finite state machine. This allows us to perform several actions on independent parts of the system concurrently.

3 Conclusions

The DEXPERT program has been in routine use since the beginning of the 1991 data taking period. It handles ~95% of the possible errors during data-taking very efficiently (i.e. correctly and considerably faster than a human expert). This has permitted us to run the data acquisition system without requiring significant expertise on shift.

Programming DEXPERT in C++ have been very successful. Specially because strong type checking removes most of the bugs at compilation time, inheritance and encapsulation facilitates test at the unit level and smooth evolution. Moreover some classes have been re-used in other applications.

4 Acknowledgments

The work presented has been developed using many building blocks of the existing Aleph data acquisition system. Without them this work could not have been done. We would like to acknowledge the Aleph on-line team, in particular J. Harvey, B. Jost and I. Videau for their useful discussions.

5 References