Title
BUILDING A PROTOTYPE EXPERT-SYSTEMS

Authors
Kalmus, D.
Hutchinson, M.
Halld, D.

Publication Date
1988-07-01

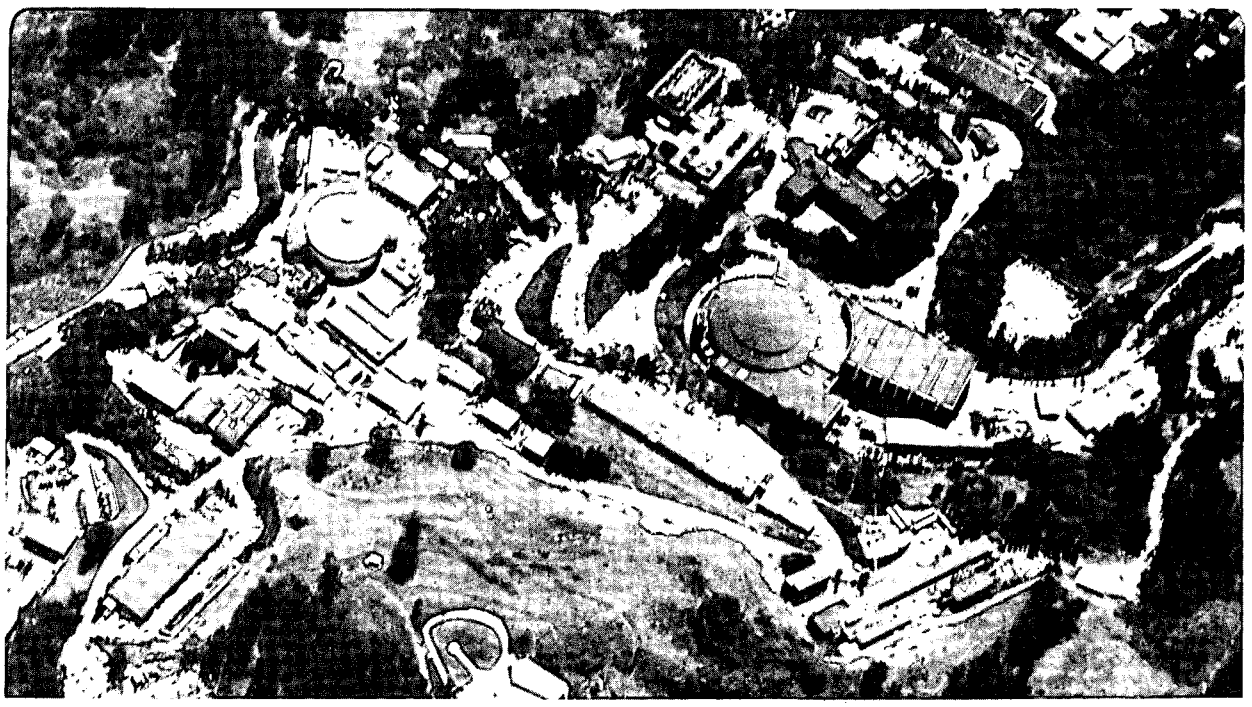
Lawrence Berkeley Laboratory
UNIVERSITY OF CALIFORNIA, BERKELEY

Information and Computing Sciences Division

Building a Prototype Expert Systems

D. Kalmus, M. Hutchinson, and D. Hall

July 1988



Prepared for the U.S. Department of Energy under Contract Number DE-AC03-76SF00098.

DISCLAIMER

LBL-25634

Building a Prototype Expert Systems

Diana Kalmus
Marjorie Hutchinson
Dennis Hall

July 1988

Lawrence Berkeley Laboratory
University of California
Berkeley, California 94720

This work was supported by the U.S. Department of Energy under Contract DE-AC03-76SF00098.

Abstract.

In the past few years expert system technology has been gaining increasing respect within the world of computer science as it offers practical solutions to problems which have previously defied computerization. This paper is the culmination of a years investigation into how LBL can practically make use of this technology to solve some of the problems being faced by its scientists.

To establish this and gain a greater understanding of expert system technology we attempted to build a prototype expert system using a commercially available expert system shell. The application we chose was to troubleshoot the hardware of the TPC particle detector (used by high energy physicists at LBL) using Neuron Data's expert system shell, Nexpert.

This paper gives some brief overviews of the theoretical and practical work done by other people in fields relevant to this project. It includes: expert systems, their development, diagnostic expert systems, and examples of expert systems built to troubleshoot electronic devices. We describe how we selected our prototype expert system and then how we went about designing and building it. For this we have detailed the knowledge necessary to start troubleshooting the TPC and the methods used to represent that knowledge within the expert system shell.

Finally we discuss the understanding of expert system technology which we have gained during this project and why we believe that this technology has a place in the future of problem solving at LBL.

Contents

I. INTRODUCTION.

1. The Project.

2. Expert Systems and Their Development.
   - 2.1 Expert Systems.
   - 2.2 Expert System Development.

II. PROJECT PREPARATION.

3. Choosing an Application for the Project.
   - 3.1 Application Requirements.
   - 3.2 Possible Applications.
   - 3.3 The Chosen Application - Trouble-shooting the TPC.
   - 3.4 Aims for a New Trouble-shooting System for the Hardware of the TPC.
   - 3.5 Resource Allocation.

4. Diagnostic Expert Systems.
   - 4.1 Diagnosing Malfunctions.
   - 4.2 Diagnostic Strategies.
   - 4.3 Model-Based Systems.
   - 4.4 Integrated Diagnostic Models.
   - 4.5 Other Electronic Troubleshooting Expert Systems.
   - 4.6 Comparison with the TPC Project.

III. KNOWLEDGE ACQUISITION.

5. The TPC.
   - 5.1 The Electronic Configuration.
   - 5.2 The Problem Outline.

6. Present Methods of Trouble-shooting the TPC.
   - 6.1 Monitoring the Data Words.
   - 6.2 Troubleshooting Using the End-of-Run Histograms and Plots.
   - 6.3 Troubleshooting During Calibration.
   - 6.4 Troubleshooting Using Test Pulsing.

7. Choosing an Expert System Shell for the TPC Application.
   - 7.1 Expert System Shell Requirements.
   - 7.2 Appraisal of Commercially Available Expert System Shells.
   - 7.3 The Chosen Expert System Shell - NEXPERT Object.
   - 7.4 EASE+ - An Application Development Environment.

8. Building the Knowledge Base for the Prototype Expert System.
   - 8.1 The Static Data.
   - 8.2 The Static Knowledge.
   - 8.3 The Dynamic Data (Input).
   - 8.4 The Compiled Pattern-Matching and Experiential Knowledge.

IV. CONCLUSIONS.

9. Summaries and Conclusions.
    9.1 Summary of Expert Systems.
    9.2 Summary of the Project.
    9.3 Conclusions.
    9.4 ACKNOWLEDGMENT.

APPENDICES.
    A. Other Electronic Troubleshooting Systems.
    B. Nexpert Questions and Answers.
    C. Programs, Data and Knowledge Bases.

REFERENCES.

I. INTRODUCTION.

In the past few years, expert systems have been gaining an increasing amount of interest in organizations ranging from scientific and technological to business, government and financial. The reason for this interest is the ability of expert systems to solve problems that had previously defied computerized solutions. Such problems are those which have no well defined procedures to reach their solution but rather involve the use of expertise gained from experience in that problem area. Expert system technology is one of the latest, and more successful, ideas to come out of the artificial intelligence laboratories; it is used to encode into a computer the expertise used by a human expert in solving a problem from within a limited field or domain.

Along with the increased interest in expert systems and a growing awareness of the types of problems they can be used to solve, a number of likely applications at LBL began to surface. This, combined with an eagerness to try out this new technology, lead to the birth of this project.

1. The Project.

Project Aim.

The goal was to investigate the ways that expert system technology can be used to solve immediate and difficult problems faced by LBL's scientists. To establish this we decided to build a prototype expert system. Furthermore, as the project was to run for just one year (July 1987 to July 1988) and our interest was in using, not developing, expert system technology, we decided to use a commercially available expert system shell to speed up and ease the construction of the expert system.

Overview.

This paper describes the work that has been done towards fulfilling the project aim. It describes how we went about selecting, designing and building an expert system to troubleshoot the hardware of the time projection chamber (TPC); a particle detector used by the high energy physics community. It gives an overview of the ideas behind expert system technology and the theory of their development, as well as some summaries of work carried out by other people in fields relevant to this project.

In the first part of this paper we will explain what expert systems are and give a summary of the current ideas on their development methodologies. The rest of the paper will follow our project through each of the phases of expert system development.

The first phase of expert system development is the preparation for the project. It includes the selection of a suitable problem or application (in our case diagnosing failures in the TPC hardware), and the setting of goals for the new expert system and its prototype (a simpler, test version). The next section gives an introduction to diagnostic systems, explaining why expert systems are used and a summary of current ideas on how they should be structured. We then give a few examples of expert systems which have been built or are being built for similar problems, i.e. diagnosing failures in electronic equipment.

The second phase of the expert system development deals with the processing of the knowledge to be incorporated into the system. This is the knowledge acquisition phase where the expert knowledge from a human expert is encoded into a computerized system. Usually a knowledge engineer acts as an intermediary, extracting knowledge from the human expert and translating it into a computer understandable form. This process accounts for most of the expert system's development time. We describe what we have learned about the device we wish to troubleshoot, the TPC, and the present methods used for diagnosing failures in its hardware. Once we knew a little about the problem, we were able to select one of the many commercially available expert system shells on which to develop our prototype. Our choice was Neuron Data's expert system shell, Nexpert Object. We then go on to detail the work that has been done encoding knowledge into Nexpert as the construction of the prototype expert system took place.

We conclude this paper with summaries of what we have learned during this project and the conclusions that can be drawn from the work we did.

2. Expert Systems and Their Development.

As we aimed to build a prototype expert systems in this project, we will start by defining expert systems. We will then go on to summarize some of the current ideas on how they should be developed, outlining the important stages in their development.

2.1 Expert Systems.

Before offering a definition of what an expert system is, we should, perhaps, first offer a definition for an expert (human):
"An expert is an individual who is widely recognized as being able to solve a particular type of problem that most other people cannot solve nearly as effectively or efficiently."
The definition of an expert system, then, offered by P J Denning is:
"An expert system is a computer system designed to simulate the problem solving behavior of a human who is expert in a narrow domain."
In addition to containing knowledge in a narrow domain (or specific field) and being capable of processing that knowledge in a human-like manner to produce a reasonable conclusion, an expert system can deal with uncertainty in its input and offer explanation for its output.

An expert system is made up of three major parts, a knowledge base, an inference engine and a user interface.

The Knowledge Base.

There are five methods of knowledge representation used at present in expert systems. The first two, rules and logic, are used to represent the conditional type of knowledge and are often combined with one of the remaining three: semantic networks, object-attribute-value triplets or frames to complete the knowledge base. All of these methods of knowledge representation are explained in the book on expert systems by Harmon and King [6] among others.

The Inference Engine.

The methods of inferencing used by the engine also vary. Hypothetical reasoning is a method where assumptions are made by the system to enable the inferencing to continue. This is usually complimented by a form of reasoning which is where assumptions can be retracted if they are found to be false. This form of reasoning is called non-monotonic. Another method, known as inductive inference, is where a minimum depth search tree is used to limit the amount of information that needs to be gathered for the systems reasoning strategies to reach a conclusion.

The method of controlling the inferencing must also be considered. This part of the inference engine controls when a rule is fired or in which order the knowledge is processed. It must have strategies for deciding where to start the inferencing process and how to move through the knowledge base during an inferencing session. There are two basic ways in which to decide where the inferencing starts. Backward chaining is a hypothesis driven approach which starts with a hypothesis and searches for data to prove or disprove it. Forward chaining is a data driven approach which means it starts with the data and searches for conclusions that can be drawn from it. Many of the more recent expert systems have a hybrid control mechanism which combines both approaches, using whichever approach is suitable for a particular part of the session.

For deciding how to move through the knowledge base, again there are two main strategies. Breadth-first searching looks at all the possible rules at one level of the search tree before the next is considered. Depth-first searching, on the other hand, looks at the next level of the search tree once, a rule has fired, before considering the rules at the same level only going back to previous levels when a path terminates and a conclusion has not yet been reached. Depth-first searching will normally lead to the first reasonable conclusion, whereas breadth-first will normally lead to all the reasonable conclusions.

2.2 Expert System Development.

In this relatively new field, a methodology for the development of expert systems is only just beginning to emerge. In this section we have summarized and combined the methodologies offered by Rolandi [9]; Cupello and Mishelevich [2]; Aikins [1] and Hayes-Roth, Waterman and Lenat [7], which detail the phases of development of expert systems. Following on from that is an outline of the problems in development which can lead to the failure of a project. The section is concluded by a paragraph explaining how this methodology relates to the way our project was developed.

The Phases of Development.

Expert system development is divided into four phases. The first phase is preliminary analysis in which a suitable problem must be selected and then the procedures to find its solution planned. The second phase is the building of a demonstration prototype from which the decision to continue the project will be made. The third phase, then, is to develop the knowledge base to a depth and breadth necessary for the solution of the problem. Finally, in the fourth phase, the system is moulded into a deliverable state where it is ready for the day-to-day tasks expected of it. By the completion of this final phase, the system must be completely integrated and delivered onto the delivery hardware.

i) Preliminary Analysis.

Selecting an application or problem is discussed in greater detail in the next chapter. Ideally this problem should be routine, moderately difficult, in an area where there are too few experts and be made up of simpler sub-problems. Project planning includes the allocation of time and people. For larger projects a development team must be created, this team should include knowledge engineers, domain experts, and software engineers to ensure technical success and to ensure user and management acceptance, members of both these areas should also be recruited onto the team.

ii) Demonstration Prototype Development.

For the demonstration prototype a sub-problem must be identified which will illustrate the power of a future system by behaving in an intelligent manner within its limited domain while being small enough to enable rapid prototyping.

This is where we arrive at what is considered to be the major bottle-neck of expert system development, the knowledge acquisition stage. Knowledge acquisition is the process of passing knowledge from the domain expert to the computerized system, this is usually done via the knowledge engineer whose job it is to encode the knowledge she has gained by interviewing the expert and/or observing him at work into the computerized system. In this stage the knowledge engineer must identify the key concepts and

relationships of the problem and consider the kind of knowledge representation to use. This will help in selecting the appropriate expert system building tools. The knowledge acquisition process is carried out iteratively, the knowledge engineer after talking to the expert, will try to formally represent what she has learned, encoding it with the tools, and then go back to the domain expert for another portion of his expertise. This goes on until those involved believe the knowledge to be adequate for solving the sub-problem of the prototype.

Because a major aim of the prototype is that it be developed quickly it is recommended that little attention is paid to the time or space efficiency of the program as this can be corrected, or taken into account in the next phase. In addition, one should aim for a simple inference engine, build mechanisms for indirect referencing (for example, using classes rather than individual instances), to keep the domain specific and general purpose problem solving knowledge separate and even at this early stage pay attention to documentation so that others may use and understand the prototype.

Once the demonstration prototype has been built, or rather the team feel it ready, it must be thoughly tested. Possible errors in input, output, inference rules, control strategies and test examples must be considered. Decisions on whether the project should continue and whether the present knowledge representation methods are to be used in the next phase must be made. If there are to be major changes it is often advised to build another prototype before commencing the next phase.

iii) System Development.

The first task to be carried out in developing the final system is to set the goals for the validation of the whole problem now to be addressed. The knowledge base for the complete system is developed by either extending the prototype or starting anew using the lessons learned from the prototype development. The knowledge base must have the depth, breadth and inferencing capabilities to cope with the whole problem domain.

The next task is to design and build the user interface and the interfaces to existing data processing systems. Finally, user training, system support and the technology transfer must be planned.

iv) Deliverable System.

Before the system can be delivered, it must be extensively tested for validity and reliability by the domain experts (both those whose knowledge is encoded and by others), the knowledge engineers, the end-users and the management. The system must be running on the delivery hardware, be fully customized and integrated before the expert system is complete.

Success or Failure of an Expert System.

Above is outlined a development methodology for expert systems, it is by no means the only one. Below, is an outline of the pitfalls which can lead to the failure of a expert system development project.

The most common problems which lead to project failure are: choosing an inappropriate application for expert system technology, choosing an application that is too large for the time allotted or experience of the expert system development team; underestimation of the costs of the project; having insufficient management commitment and, finally, having difficulties with user acceptance.

For successful prototype expert system development, it is advisable to have a development team of at least two full time staff, headed by someone who is technically trained, with proven management skills and training or experience in AI. Such a project is estimated to take about a year with a budget not to exceed $250 000.

Developing This Project.

To achieve the project aim we are only building a demonstration prototype, therefore, it is only necessary to complete the first two phases of expert system development, the preliminary analysis and the development of the prototype. The first of these phases, the preliminary analysis, including the selection of an application, the selection of a sub-problem (from that application) for the demonstration prototype, and the project planning are described in part II of this paper, "the preparation of the project." The demonstration prototype development phase encompasses the task of knowledge acquisition and involves the knowledge engineer learning about the TPC detector and the present methods used to troubleshoot it, choosing an expert system shell which is able to represent that knowledge and, then, in encoding the knowledge into that shell. These are discussed in the part III of the paper, "the knowledge acquisition." Also within this phase of development is the evaluation of the prototype, this will be discussed in the final part of the paper with the project conclusions.

II. PROJECT PREPARATION.

This is the preliminary analysis phase of the expert system development. It is here that we explain how we chose the problem of troubleshooting the hardware of the TPC. First we have outlined the requirements of a successful expert system application and, after considering a number of possible problems, chose the one which best fitted those requirements. Once the application had been selected, the goals for the project were set and the resources were allocated. The type of system we were designing was a diagnostic expert system. The latter section, then, looks at some of the work done with diagnostic systems, the development, and some examples of other electronics troubleshooting expert systems which are being built or have been built recently.

3. Choosing an Application for the Project.

3.1 Application Requirements.

To find an application that will fulfill the project aims, we must first establish the types of application that are suited to an expert system.

Feasibility.

For the project to be feasible the application must be useful and cost effective. It must be a labour-saving system that does not involve huge effort to build as there is a limited time for this project to be completed. The application, because of this time constraint, must be ready to build now with all the expertise easily available (although not abundant, as then an expert system is obsolete) and the human expert sympathetic to the project, it should not be an exercise in persuading reluctant individuals to part with their expertise.

The Knowledge Domain.

The application's field of knowledge (domain) must be both narrow and deep. The domain must be fairly narrow and not involve the use of 'common sense' to make it possible to encode into a knowledge base. It must be deep to make the project worthwhile, if the domain is both narrow and shallow it is probably too simple a problem to solve using an expert system.

An ideal problem for an expert system application needs knowledge that is well-formulated (but ill-structured), in other words it must be based around facts and production rules, not algorithms (as in procedural programming). The paradigms that are particularly suitable to expert system development are: trouble-shooting (diagnosis/prescription), planning or constraint satisfaction, and design or discovery paradigms.

Expert systems are well suited to evolutionary development of their knowledge bases, this method of development is not a requirement but can be very useful for some applications.

The Task.

An expert system application must be made up of tasks that are largely intellectual skills that is easy to verbalize, they must not involve manual (e.g. glass-blowing) or sensory (e.g. visual) skills as these cannot be easily encoded in an expert system.

The second task requirement is that it must be solved in a reasonable length of time, somewhere between 10 minutes and a few hours. If it is soluble in less than 10 minutes it is probably too simple a task to be solved using an expert system, likewise, if it takes more than a few hours to solve it is probably too hard (at present).

Finally, the task must have no algorithmic solution, for if it does then conventional programming techniques should be used. The reason to favor conventional computerized systems is that are much more reliable and easily testable than expert systems.

3.2 Possible Applications.

We initially started looking at five or six different application, before narrowing it down to three; automatic *discoveries* in particle physics, a run management system and the detection and analysis of hardware failures in the TPC particle detector. The first

the detection and analysis of hardware failures in the TPC particle detector. The first two were later rejected in favor of the final application, which were found to satisfied the requirements for both an expert system and the project the best.

3.3 The Chosen Application - Trouble-shooting the TPC Hardware.

This application will be discussed in detail later, at this point reasons are given for the selection of this application.

Feasibility.

An expert system would make the TPC easier to use, taking less time to trouble-shoot and calibrate, these factors combined with the fact that the trouble-shooting is, at present, done by highly skilled scientists and technicians, make an automatic system cost effective. The expert for this problem, Dr Alan Clark, is sympathetic towards the project, although he is often very busy (hence the need for this system) and so has very little time to spare, making the knowledge acquisition more difficult.

The Knowledge Domain.

Trouble-shooting the hardware of the TPC uses a domain of knowledge which is narrow, dealing only with the workings of the TPC, but also has a deep knowledge of the problems, spanning from surface rules to more fundamental rules about the operation of the hardware.

It's trouble-shooting paradigm is well suited to expert systems because it is well formulated and ill structured, it is, indeed, built from rules and facts and its method is not structured, it's path through the knowledge base is not predetermined and is different for each session (given different data).

It would be considerably more convenient to develop the knowledge base in an evolutionary way, developing first a working prototype and then building on that.

The Task.

The task of trouble-shooting the TPC, again, fits well with the requirements, being largely an intellectual skill, although it does sometimes involve using visual skills to look at graphs and plots. This problem could be overcome by using the original data (the plotting is done for the ease of the human expert). The solving time of the task lies well within the acceptable boundaries, for most of the failures.

In this application, the task has no algorithm for it's solution so does not lend it's self to conventional programming techniques. Although a highly reliable system is desirable, it is not necessary for this application, it is not a disaster if the system, occasionally, cannot correctly diagnose a fault.

3.4 Aims for a New Troubleshooting System for the Hardware of the TPC.

The aim of our project was to automate the detection and analysis of the hardware failures in the TPC. At present, this is largely done by Dr Alan Clark, it is a time consuming exercise and the relevant expertise is very scarce. Many of the people who make use of the TPC are either infrequent or novice users, therefore, it would be beneficial to have a comprehensive, easy-to-use trouble-shooting system for the hardware.

Specification for a Full System.

More specifically a new system should be able to notify the user of failures, giving information such as what has gone wrong, where and how to fix it, having asked the user the minimum number of questions, and being able to explain its reasoning if requested. Ideally the system should be designed for use by both the expert whose knowledge is encoded in it and by a novice user. It should be able to ask for a specific area to be focused on if a particular failure is suspected by the expert as well as being able to look for failures when the user has no idea what is wrong. Reasoning behind a conclusion must be explained by the system only if it is required, so as to enhance an experts understanding and not confuse a novice to the TPC.

Specification for the Demonstration Prototype System.

In order to build our prototype we identified a sub-problem of a size that we considered manageable for our year-long project. The prototype system is being designed to cope only with the wire channels (not the pads), this narrows the number of channels from the original sixteen thousand to nearer twenty-two hundred. To further limit the problem, just one of the many troubleshooting methods was chosen: the use of the test pulsing program and the analysis of its output (explained later in section 5.2 Present Methods of troubleshooting the TPC).

These choices were made in consultation with the expert taking into consideration the prototypes ability to show the potential strength of an expert system on a useful sub-problem.

3.5 Resource Allocation.

There was to be just one person working on this project full-time (Diana Kalmus) and the expert (Alan Clark) was consulted for a couple of hours each week. The time allotted to this project was exactly one year.

4. Diagnostic Expert Systems.

Before continuing with the development of our system, it is worth pausing to consider the type of expert system we are to build, and review some of the work done in this field. We are building a diagnostic or troubleshooting system which, at present, accounts for the largest application area of expert system technology, it is the most popular paradigm used. The first diagnostic expert system built was MYCIN in 1976 at Stanford University. It was used to diagnose bacterial infections of the blood. Now diagnostic expert systems range in their fields of application, from medical to mechanical to electronic. It is in the electronic domain that we will focus in this paper.

Overview.

In the following section we discuss some of the recent work in diagnostic expert systems, particularly in the electronic domain. First, we have summarized a couple of new ideas put forward in this field and then explain how they are being put into practice. The first idea consists of outlining the stages taken to diagnosis a malfunction; this is used to help the knowledge engineer to gain a clearer view of the problem. The second of these ideas outlines the various diagnostic strategies which are used in designing a system, either singly or in groups depending on how deep and complete the knowledge must be. Putting these ideas into practice gave birth to the concept of model-based expert systems, which are designed to incorporate a deeper level of knowledge. The final sections cover some diagnostic expert systems which have been built and demonstrate how the theories of model based diagnostic expert systems have been put into practice.

The new concepts, ideas and methodologies of diagnostic expert systems, that we have summarized in this chapter were put forward by Fink, Lusth and Duran [17]; Merrit [21] and Milne [22].

4.1 Diagnosing Malfunctions.

The goal of a diagnostician, human or machine, is to offer explanations of the observable facts. To do this successfully it must have knowledge of the set of possible malfunctions along with the relations between them and the observations that can be made about them. There are five steps taken in diagnosing a malfunctioning system:

1. Observation of a malfunction and the gathering of input data.
2. Generation of a plausible set of hypotheses.
3. Determination of the evidence needed to accept or reject the hypotheses.
4. Testing to narrow the sets (preferably ordered such that the most useful or cheapest tests are carried out first).
5. Acceptance or rejection of the hypotheses until the malfunction has been identified, fixed, and the system is functioning normally again.

In addition, the diagnostician should terminate the search when the cost of searching for the failure outweighs the cost of replacing the malfunctioning unit.

4.2 Diagnostic Strategies.

There are four basic levels of knowledge representation for diagnostic expert systems which range from the top level compiled pattern matching knowledge which is shallow in nature and based on experience, through to the functional knowledge, then to the behavioral knowledge and finally to the bottom level structural knowledge which is deeper, more basic knowledge.

Starting at the bottom level, structural information is information about the connectivity of a system. It is useful to know this if more direct knowledge is not available. When a failure occurs in a device, the structure of that device changes in a way that corresponds to that particular fault. For example, in an electrical circuit containing a light bulb, the circuit must be continuous for the bulb to light, if the structure is altered and there is a break in the circuit, a failure occurs causing the light not to work. So, if the new structure can be matched with either one in a fault library or with a simulation of how a hypothesised failure would look, then the fault has been identified. This approach to diagnosis is known as the "faulty model".

The next from bottom level is behavioral information. Behavioral information is that which is made up of the states of components under certain conditions. In this situation "qualitative simulation" is usually used as it restricts the possible set of values by changing numerical values to high, low, or normal for example.

Functional information describes what a component is used for (gives its function). This is at a higher level than behavioral information which would tell you, for example, that a components output was higher than its input, the functional information associated with this component would tell you that it is an amplifier.

The highest or shallowest level is the compiled pattern matching information. This is the information which has been gained by experience and is what distinguishes between a novice's and expert's performance and takes the form of rules-of-thumb, heuristics or short-cuts. An example of experiential knowledge, going back to the light bulb circuit, is: "if the light does not work, the most likely fault is that the bulb has blown". Although any break in the circuit would cause such a failure, an expert will use rules such as this, which he has gained from experience, to cut down on the diagnostic effort.

Expert Systems can be built at any of these levels. The earlier expert systems were built using only top level, shallow knowledge. More recently expert systems have been designed using many or all of these levels to take advantage of both the shallow knowledge for solving frequent problems, and the deeper knowledge for solving new or infrequent problems successfully. Systems have been built using only the lower level or deeper knowledge but these are inefficient at solving frequently occurring problems and so this is not an ideal design.

4.3 Model-Based Systems.

In the past these diagnostic systems have been strongly rule-based, using shallow, experiential knowledge to reach their solutions. The difference between this type of system and the human expert is that it degrades badly at the edges of it's knowledge. It has no deeper understanding with which to reason from first (or at least earlier) principles as its human counterpart will do when faced with an unfamiliar situation. Worse of all is the fact that a purely rule-based expert system is unable to tell the user when it is at the edges of its knowledge and, therefore, much more likely to reach the wrong conclusions. Recently, in an effort to combat these problems and map the expert system's knowledge representation more closely to that of the expert it intended to mimic, model-based diagnostic expert systems have been developed. These model-based systems contain both shallow, experiential, specific knowledge and deeper, general knowledge of the domain allowing for more graceful degradation at the edges of its knowledge.

A major advantage of using expert systems for these model based systems is the visibility of the knowledge and the explicitly represented structures. Components can be logically connected and their associated facts, operations, rules and control procedures are grouped around them. Graphical interfaces are becoming increasingly popular in these modern expert systems for the development of the system and displaying the

results, conclusions and advice to the end-users.

4.4 Integrated Diagnostic Model (IDM).

The Integrated Diagnostic Model is a good example of a general diagnostic expert system design developed by P.K.Fink, J.C.Lusth and J.W.Duran [17] [18]. The IDM is comprised of two separate knowledge bases: the experiential, which contains the shallow, compiled pattern matching knowledge and the physical, which contains the deeper levels of knowledge. An inference engine is used to link the two to give one coherent and complete knowledge base.

The physical knowledge base is the "model" part of the system, combining the structural, behavioral and functional knowledge to ensure a thorough understanding of how the device works. The components of the system are arranged in a hierarchical structure, where the system is defined in terms of a number of sub-systems and how they are positioned and connected, then the sub-systems are defined by their constituent components and so on recursively. This structure helps the system to focus on a problem area, and mimics the method that we use in trouble-shooting anything but the simplest device. For example, if the radio in your car stopped working, you would examine the electrical sub-system before you looked at the mechanical sub-systems.

4.5 Other Electronic Troubleshooting Expert Systems.

In the last couple of years a number of expert systems of the same paradigm (troubleshooting/diagnostics) and in similar domains (electronics) to this project have been developed. Here we briefly describe three such troubleshooting expert systems that have been developed elsewhere.

FIS (Fault Isolation System).

US Naval laboratory's FIS developed by Pipitone in 1986 [23], using Franz Lisp, is an expert system designed to diagnose faults in analog electronic systems. It aims to isolate failures in the device to the level of larger components such as an amplifier or power supply. It is a generic electronic troubleshooting expert system which can be modified for each device by the knowledge engineer using the schematic and block diagrams and other information supplied in the documentation of the device. FIS provides a library of common electronic components and component properties to minimize the effort of knowledge acquisition for a new device. Knowledge acquisition involves describing the connectivity and any unique components and properties.

FIS is efficient at knowledge acquisition, it reasons quantitatively with probabilistic reasoning methods developed especially for that device and troubleshoots from a functional model of the device.

TEST (Troubleshooting Expert System Shell).

Carnegie Group's TEST developed by Kahn et al in 1987 [19], using Carnegie Group's Knowledge Craft (a general purpose expert system shell) is a domain-independent diagnostic problem solver. Its approach is half way between the causal-model and the rule-based model, using a weak causal model to describe the causal links between failure modes and the diagnostic rules which constrain and direct the reasoning. It includes a library of module types such as failure-modules, test-modules, etc. from which to build the diagnostic system.

AI-TEST.

AI-TEST, developed at Tel Aviv University in collaboration with Intelligent Electronics Inc. and by Ben-Basset et al in 1988 [13], using C, is also a generic electronics diagnostic system building tool. Its developers wished to build an expert system that would be able to troubleshoot a wide variety of devices with a minimal amount of effort to train it for each specific device. It aims to provide an intelligent service manual that guides a user through the diagnosis of a device by detecting and locating a fault down to the smallest replaceable component. The design of AI-TEST allows it to handle large scale device's and cope with both analog and digital data. It uses a functional level model to allow the expert system to reason from first and second principles.

Summary.

As one can see from these examples, expert systems to troubleshoot electronic devices are being written using expert system shells, AI languages and even conventional languages. In spite of this diversity, these systems show some similarities. They all aim to reduce the Knowledge acquisition effort. To do this using some kind of model or object oriented knowledge representation making it easier for the knowledge engineer to represent the domain knowledge in a way that closely matches how the expert reaches his solutions.

4.6 Comparison with the TPC Project.

We have discussed some of the recent work done with diagnostic expert systems, focusing particularly on the troubleshooting of electronics. Many of these ideas were helpful as the design of our system took place. However, the major difference between the examples described here (more detailed descriptions of these as well as a listing of further reading are given in Appendix 1) and our problem is the complexity and repetition of the TPC's design, making a model hard to build.

The examples all modeled devices with a very limited number of circuits, the TPC has sixteen thousand almost identical circuits (or channels). To further complicate matters, these channels are related to each other in a variety of ways, for example, by sharing a component, a power supply, adjacent physical space and so on. All these relationships must be represented which posed quite a problem as we did not want to explicitly represent all sixteen thousand channels and their relationships. Several attempts, which are described later in this paper, were made to overcome this problem. The system we are building uses a model to represent the components that lie along each channel but rules are used to represent the relationship of the channels to each other.

III. KNOWLEDGE ACQUISITION.

The knowledge acquisition phase is the transfer of knowledge from the expert to the computerized system and is usually done by a knowledge engineer who learns about the problem from the expert and then encodes that information into the computer.

The knowledge engineer's first task, then, is to learn about the problem by interviewing or observing the expert at work. For this project the knowledge transfer took place in weekly interviews, mostly between November 1987 and February 1988. The information gleaned from these sessions with the expert is in the next two chapters; the first describes the TPC's electronic make-up and the second describes the methods presently used to troubleshoot it.

The second task within knowledge acquisition is the choice of language or shell within which to build the new system. We had already decided, for this project, to use a commercially available expert system shell so our task was slightly narrowed. The choice of shell should be made once the knowledge engineer has determined the type of knowledge representation necessary. So, following the TPC description, the next chapter explains how we chose an expert system shell for this project; it outlines a number of competing shells and then gives a description of the one we selected with the reasons for selecting it.

The Final part of knowledge acquisition is the encoding of the knowledge into the machine. This we have described in some detail in the last of these chapters.

5. The TPC.

The TPC detector is a large gas filled cylinder, 2m in length and 1m in radius. Its function is to provide 3 dimensional images of e+ e- collisions which occur within the cylinder. At both ends of the cylinder lie circular planes, each divided into 6 sectors (a total of 12) that contain all the detection equipment. On each sector there are 184 proportional wires, which are used to give the radial data, and 15 positional pad rows, which give azimuthal information. Both the wires and pads give ionization information and make up a total of 16000 channels in the TPC detector.

5.1 The Electronic Configuration.

The Computers Used.

The TPC detector runs on several computers concurrently, the PDP 11/70, the VAX 11/780 and the VAX 11/782 all connected by DECNET on site at SLAC. The PDP 11/70 does the majority of the monitoring, through a dedicated PDP 11/04 (not connected through DECNET).

The Path of a Channel.

For each channel there are a number of components connected in series, the pre-amplifier is located in the detector, the feed-through ring is situated on the boundary of the detector, others are in an adjacent electronics house; these include the shaper/amplifier, the charge coupled device (CCD) board, the digitizer (these three are always grouped together, occupying the same bins in adjacent racks), the readout master control and the line driver. Finally, situated remotely in the control room is the large data buffer (LDB).

The Function of the components.

The pre-amplifier amplifies the signal which was received on the wires or pads. The feed-through ring is needed to pass channels from the high pressure interior of the detector to the outside.

The shaper/amplifier amplifies the signal so that it may pass through the electronics with the minimum of distortion. The CCD board is an analogue memory which is capable of storing a signal received from the amplifier every 100 nanoseconds, with a maximum storage capacity of about 400 buckets of information for each of the 16 channels on that board. The digitizer works at a much slower rate than the CCD, it only digitizes the signal when it is triggered; the trigger system as well as activating the digitizer, slows down the CCD output to a rate that the digitizer can cope with. The readout master-controller is used to switch the channel or board on or off. The line driver is the third of the amplifiers (this time a digital one), and is used to boast the signal for its journey to the remote LDB.

The LDB is made up of 38 lists, (plus an extra list loaded from another part of the detector) each of which have varying numbers of channels connected to it. The number of channels depends on the expected output and are arranged so that processing is not held up by overloading of one list enabling near parallel processing and at the same time trying to keep the number of lists to a minimum.

The Arrangement of Channels.

The pads are arranged in groups of 4, 8, or more usually 16 channels, groups of less than 16 are combined to make 16. In these combination groups channels may be from

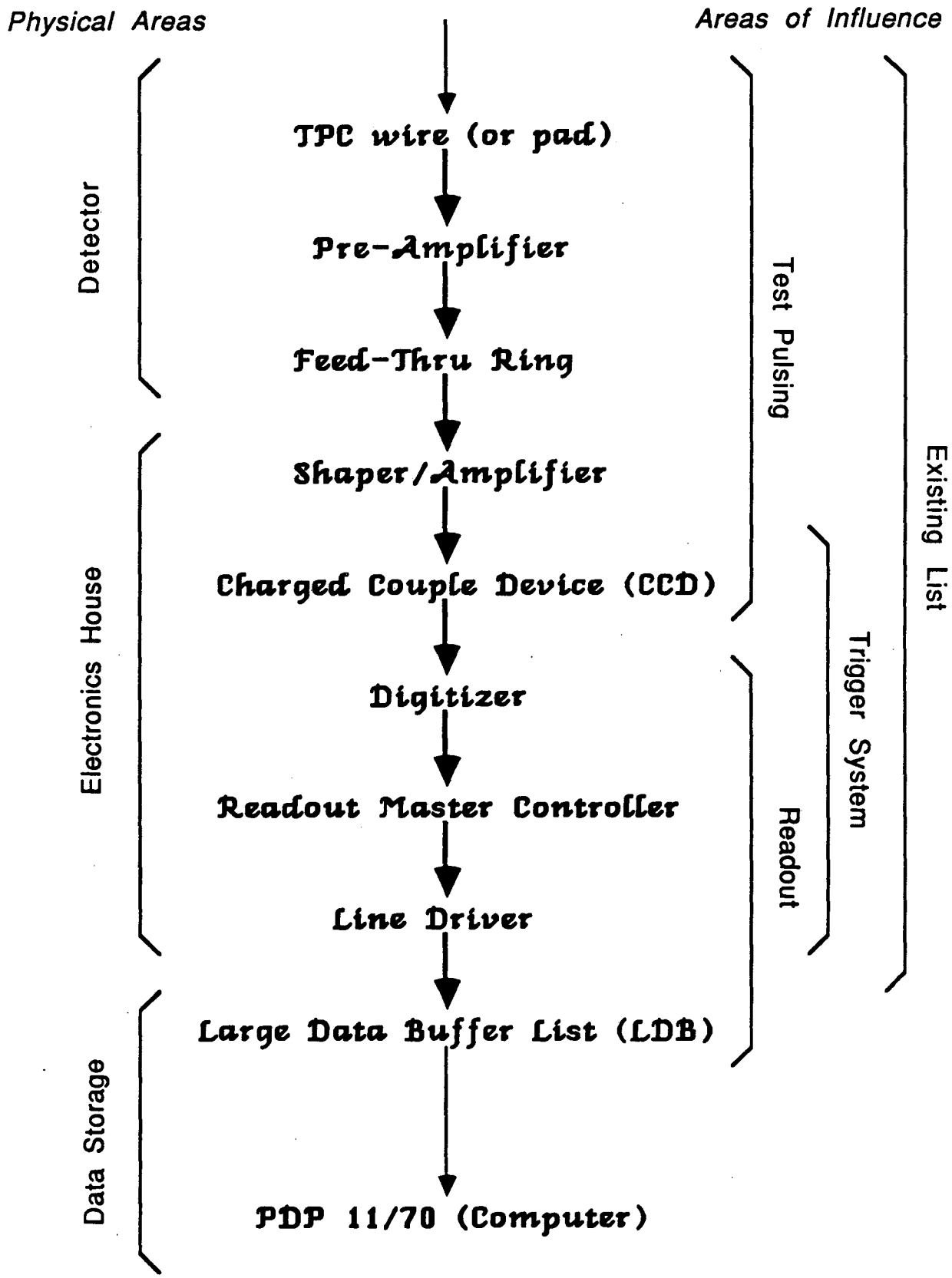


Figure 5.11: **The Path of a TPC Channel.**

adjacent sectors, rather than being all from the same sector as usual. The wires are arranged in groups of 16 adjacent wires, there are 12 of these groups on each sector (the last one being only partially filled). All these groups share, among themselves, the same shaper/amplifier, CCD and digitizer boards. The groups are combined into lists within the LDB. There are 12 wire lists and 18 pad lists; each wire list is contained within 1 bin, pad lists, however occupy between 2.7 and 4.0 bins.

For monitoring the current state of the detector there is a monitor system. This monitors the temperature, pressure, high voltages and magnetic currents displaying available information at a terminal. The monitor system is linked to the LDB where it has its own list.

The Addresses of a Channel.

The pre-amplifier is positioned near the wire or pad it corresponds to in the detector. At the feed-through ring a channel undergoes a change in it's address, from the physical location it occupies in the detector to the position of it's electronic components in rows, racks, bins, boards and channel-number; in addition, the feed-through itself also has an address different from either of the above. The shaper/amplifier, CCD board and the digitizer are always grouped together, situated in identical positions on adjacent racks in the electronics house.

5.2 Problem Outline.

The problem is to detect, analyze and advise the actions necessary when any hardware failure occurs. It is possible for any component on any channel or group of channels to fail, this failure must be detected, located and fixed. A group of channels could be a board, bin, rack, row, list, sector, endcap or even the entire detectors worth of channels.

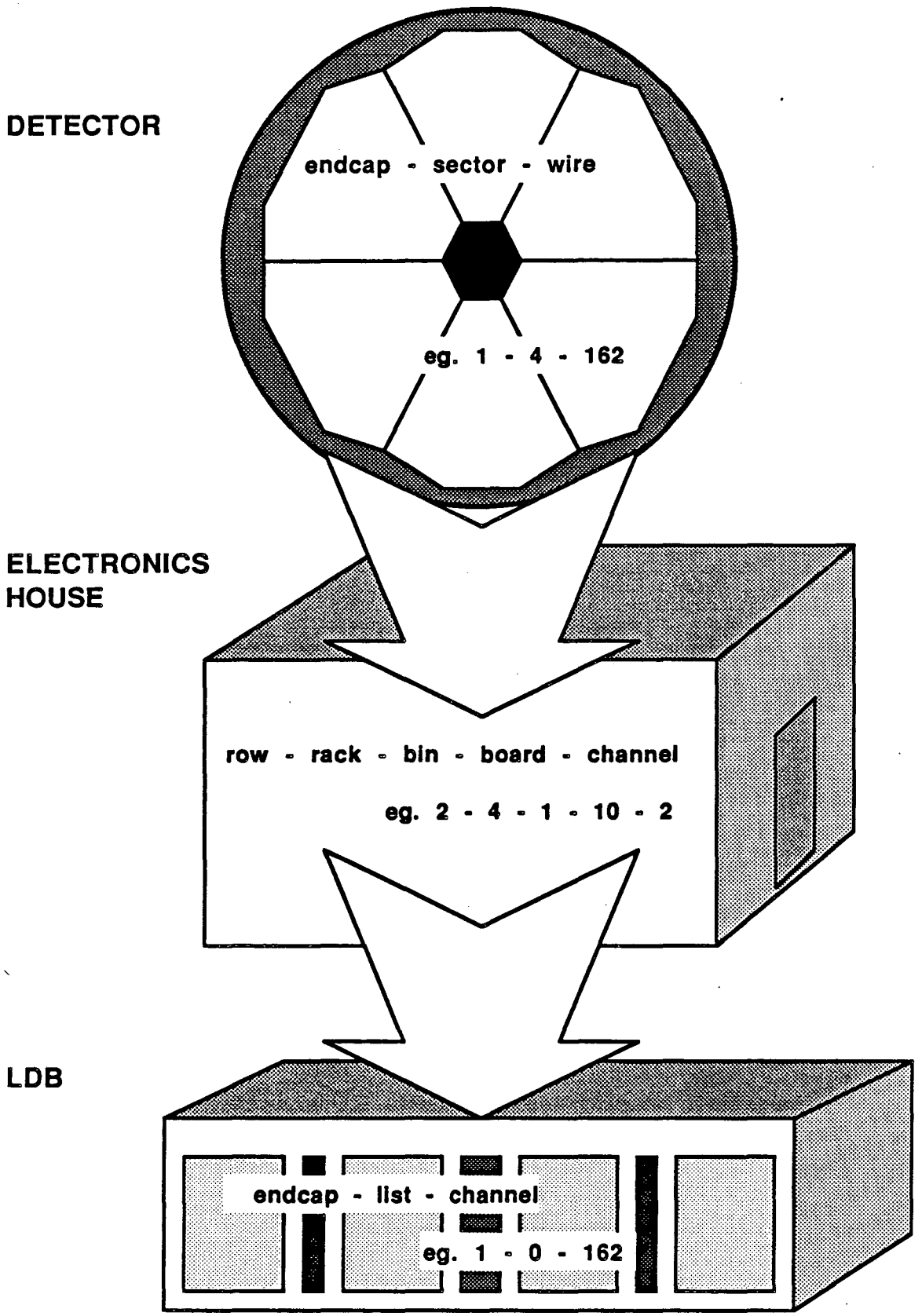


Figure 5.12: **The different addresses of a Channel.**

6. Present Methods of Trouble-shooting the TPC

In this Chapter we describe four methods used at present to troubleshoot the hardware of the TPC. The first of these methods diagnoses malfunctions while the detector is running. It uses a Fortran program called WATCH to monitor the number and addresses of the words streaming in from the detector. The second involves searching the scatter-plots and histograms which are produced at the end of each run for channel failures. The third method for troubleshooting takes place during the calibration of the detector. The final method, test pulsing, is designed especially for troubleshooting and can be used only when the detector is not running (accumulating experimental data). Before continuing with this chapter it is worth mentioning that the figures used are not to scale, but are just used to help explain some of the programs.

6.1 Monitoring the Data Words.

Two types of failure can be found by monitoring the detector with the program WATCH. They are the level of noise present on a channel and the order in which the channel data is read.

A channel becomes "noisy" when its pedestal within the CCD has drifted down, allowing more data through than is desirable for the analysis. WATCH continuously counts the number of words on each list, the word counts form a rough pattern which the operators come to learn. When there is a disruption in the pattern, for example continuously high values (indicating a noisy channel), the operator must try to identify and correct the failure. When trouble is suspected a program can be called to collect histograms for each noisy channel. Most noisy channels can be made quieter automatically by raising the pedestal, although care is needed to ensure it is not raised too high as this results in a loss of useful data.

The second of these types of failure is the ordering of the data words. A word is made up of a pulse-height, channel-address and a bucket-number (a bucket is a chunk of data). Normally the channel number increases monotonically within a given bucket-number and will reset when the bucket-number increments. If a word appears out of sequence there is a failure caused by one of the following possibilities: the data have been transposed due to zero suppression, so that the data are correct while the order is not; there has been a short circuit between the boards address, making it difficult to tell where the error occurred; and finally, a bit has gone bad, giving the wrong channel address. The latter is the most common and can be hard to detect when it only occurs occasionally. It is difficult to tell which of the two non-sequential bits is in error. Has the first gained a bit or the second lost one?

6.2 Troubleshooting using the End-of-Run Histograms and Scatter-Plots.

A histogram is produced for the wires and a scatter plot for the pads for each of the 12 sectors. These end-of-run histograms and scatter-plots are used to find missing channels (single channels are difficult to see on the scatter-plots) or groups of channels which belong to the same board or are connected to the same power supply etc. Missing channels usually indicate that there are bad connections somewhere in the channels path or in the power leading to any of its components. However locating the problem

connection can be difficult as the power supply, boards, bins etc. are all configured differently within the detector. When a failure is detected, previous plots must be looked at to establish whether the fault is a new one or if it was already there.

6.3 Troubleshooting During Calibration.

Calibration is carried out by CalTPC, a fortran program (run on the PDP 11/70). The program is divided into two major parts, the calculation of the pedestals and lower limit RAMs (LLR) and the test pulsing sequences.

Setting the Pedestals and LLRs.

The Pedestals are the level below which data is not read and the LLRs are the level above which the data is assumed to be significant (not noise on the channel). CalTPC starts by turning off all the channels, after asking the user which list is to be calibrated and then whether to refine the process further by specifying a board or bin within that list, the program then sets the lower limit RAMs (LLR) of the specified channels, four at a time, to zero. This allows all the data from the channels to be read out, giving a known number of words per channel (a constant word count). The channels are then tested four at a time. This is where the faults may be encountered. Deviations from a normal word count indicate problems with the digitizer board, read out control, LDB or power supply. Determining which of these components is in error depends on the type of deviation and the number of channels which deviate. There are a number of rules which can be used to lead to the identification of the problem.

There are other possible errors to be found in the LDB at this stage, even when all the read outs are correct. These can be found by looking for parity errors which can be a result of the LDB running too slowly. This is difficult to find using computer simulated data because the computer is not as fast at sending data as the detector is. So these errors are found by using a 'spy' program while the detector is running. It is important to first correct the read out master controller problems because these can also cause parity errors in the LDB which are much harder to find.

Next the program plots the pulse height versus bucket number (or time) for each channel fitting a straight line to it as shown in figure 6.31. Measurements of slope, pedestal level (intercept) and noise (RMS deviation) are taken to calculate the value of the LLR, then all four values are output by the program as a single line of text. The program then high-lights the values lying outside pre-determined ranges. There are a number of rules which the user applies to determine problems in the shaper/amplifier, CCD and digitizer boards or power supplies.

Test Pulse Sequences.

Test pulsing is used mostly when calibrating but can be used at any time for finding problems on suspect channels (although, at times other than calibrating another program, DeadCh, is used to test pulse the TPC). The program sends test pulses to four channels at a time, it pulses the channel many times at each of the sixteen different pulse heights that are tested. Test Pulsing involves sending signals or pulses of specified height and width to the inputs of the components along the path of the channel. For calibrating, CalTPC uses only the input to the wires and pads themselves, enabling testing of the entire channel; this is done by sending an electrical pulse to a grid which sits just in front of the sectors, within the detector. This program pulses the entire endcap although it is possible to specify the pulsing of just a single sector. However, although the test pulse cannot be sent to individual channels, it is possible to only monitor (enable) those channels of interest.

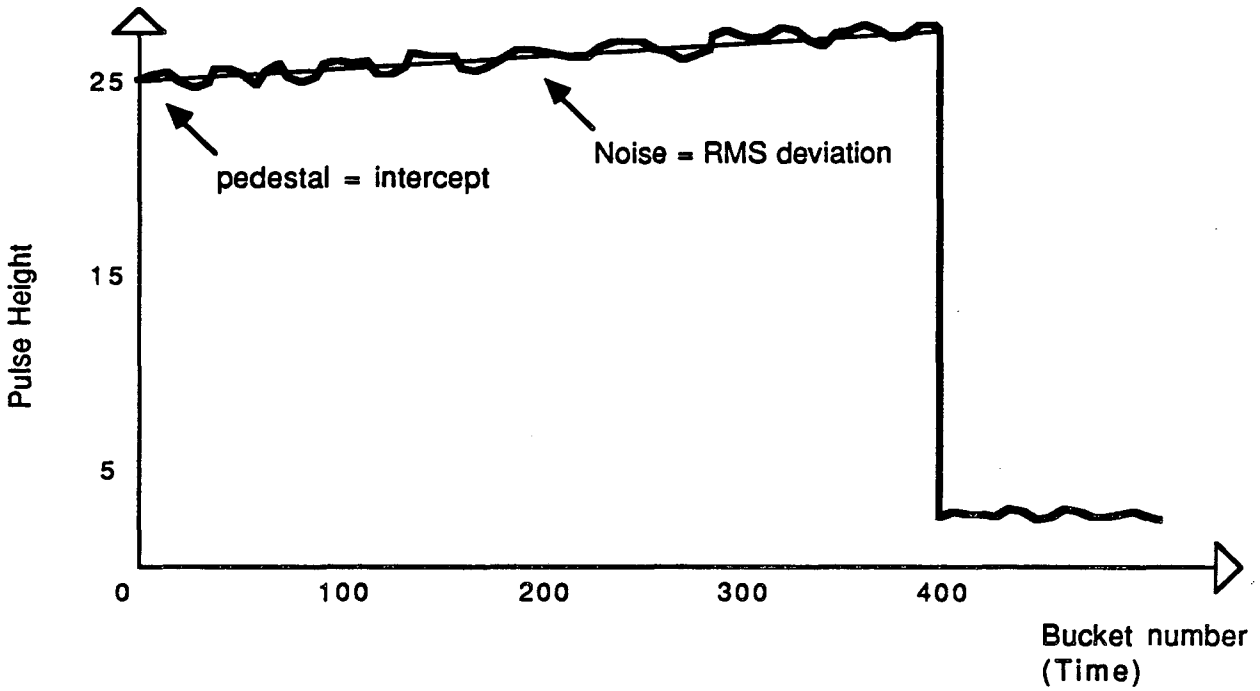


Figure 6.31: **CalTPC plots used for calibrating the LLRs**

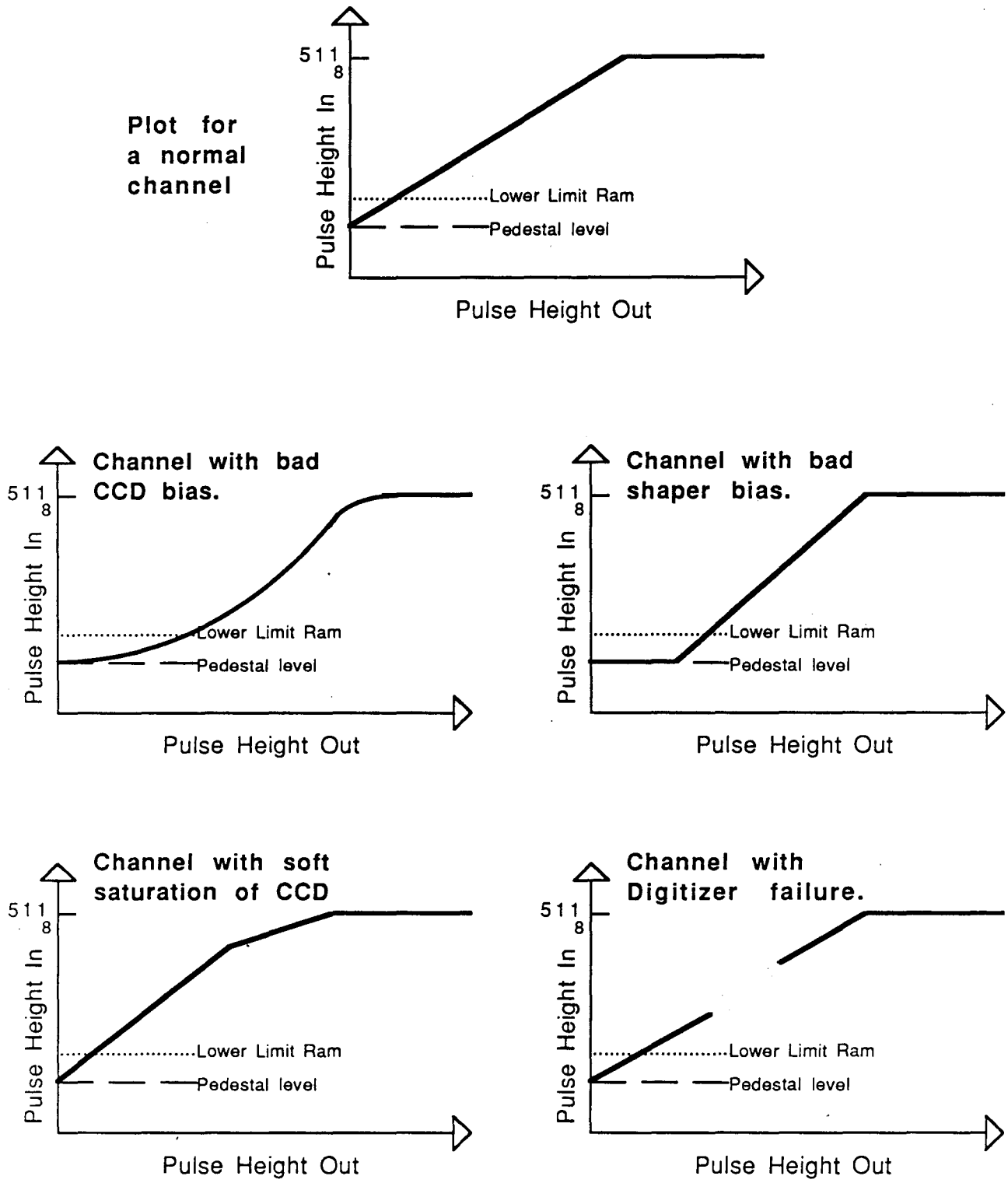


Figure 6.32: **Examples of plots produced by GainCal.**

The program, GainCal, is then run to analyze the data collected in CalTPC. It plots a graph of pulse height in against pulse height out for the test pulse sequence data for each channel of interest and then fits a curve. From these plots the calibration values are calculated and placed in a database for future reference. It then checks to see if the channels are working correctly and highlights any that it finds are not. It is then necessary to locate and fix the failures on these channels. To help with this a third program, GainDis, can be used, it displays the information gathered and calculated in the program GainCal in graphical form. Some examples of these graphs are shown in figure 6.32 which shows the plot of a normal channel and the plot that indicate certain types of failure. From these plots it is possible to determine the type and location of the problem. Failures which cannot be fixed immediately have error codes attached to them in the fitting program and the lower limit RAM so that their data are ignored during analysis and can be fixed at a later date.

The calibrating program, CalTPC, is a CPU intensive and slow program, using about 80% of the CPU and taking roughly 20 minutes per bin to run. It takes 11 hours of continuous running to go through all the bins in the TPC once, and usually the program is halted every few bins to deal with the failures that may have arisen. All in all, calibrating takes between two and four days of highly skilled persons time, depending on the number of failures. There is a much faster program, CalChk, that is used just to check the calibrations. This program tests only 4 different pulse heights, and then checks to see if they compare well with the values stored in the database from previous calibrations. If the match is bad then the channel is either test pulsed using an alternative test pulsing program if a failure is suspected or it is re-calibrated.

6.4 Troubleshooting using Test Pulsing.

The usual reasons for test pulsing are that the detector has been altered or changed in some way, when the system has been down for a while or when a failure has been found while doing something else.

The Test Pulsing Program.

This program, called DeadCh, searches for dead channels in the TPC by test pulsing them individually. It takes five measurements and then displays these along with the full electronics house address. As the program commences it asks the user to choose whether to test pulses by List or by Sector. Test pulsing by List is the usual choice and is used when checking after a down period. Test pulsing by sector is chosen if the TPC has been physically changed, the TPC is open and something is wrong, a broken wire is suspected or (because of the way this program is written) when you want to examine just one channel or a group of channels that do not make up complete boards.

The next choice faced by the user is where to send the test pulse, there are three choices at present, to the grid which sends an input pulse to the wires (and pads) and therefore test pulses the entire system, to the shaper/amp input or to the input to the CCD. There is a fourth position that the hardware of the TPC would allow a test pulse, at the input to the pre-amps, but this has never been used and so the system is not yet debugged. The default for the test pulse source is the grid because it is the one that tests the whole system or path of a channel. The source is changed when a fault has been found using the grid test pulse (repeated at least once) and one is trying to narrow down the possible area of failure. There are only two reasons for initially starting to test pulse on a source other than the grid, they are that the grid is out of action or when a particular problem is suspected.

Interpretation of the Output.

The output for the DeadCh program is stored in an ascii file called DeadCh.FUL for each session. In addition the DeadCh program provides the input for MonChn which graphically displays the data on the troubleshooters request. Both DeadCh.FUL and MonChn are used to analyze the operation of the channels that have been test pulsed by the DeadCh program. MonChn plots a histogram of the pulse height of a single channel against its buckets of data for a single test pulse. Figure 6.4a and 6.4b show the plots produced by MonChn for channels that are working normally when test pulsed at the grid or shaper/amp and CCD levels respectively. They also show how the values output into the DeadCh.FUL program are calculated. These five values are the results from test pulsing a channel and are explained in turn.

The Pedestal is the pulse height level when no signal is present. The actual level of this is adjustable for each board, this means that the variation between channels within a board should be small, while they may vary significantly from board to board. The normal range of values for pedestals (where they are assumed to be working normally) is between 10 and 50, these values are independent of pulser source and are only affected by changes to the digitizer or CCD.

If the pedestal values within a board are outside of the normal range, it indicates a bad digitizer or CCD board or a bad connection between the two. When a pedestal has the value of 1 it usually means that the value is less than 1 as that is the lowest reading which can be shown. So a value of one, probably meaning a negative number, indicates that there is a bad capacitor on the CCD board. When the ailing pedestal values are bin rather than board wide, it usually indicates a failure in the CCD bin such as power failures or clock signal failures. Another possibility is that the temperature of the electronics house is too high (pedestal is too high) or too low (pedestal is too low).

The Edge is the place where the test pulse is first seen, the first edge of the peak in the plot. This edge is normally well defined as the test pulse should produce a sharp peak. The value of the edge is dependent on the source of the test pulse, at the grid the edge is 2 buckets later than when the test pulse source is either the shaper-amp or the CCD where 45 and 46 are considered normal values. When the edge is outside of the normal range it can indicate one of three possible problems; an interfering program is being run somewhere else in the system, there is a hardware failure in the test pulsing system, or the channel is oscillating.

The Pulse Height is not only dependent on the source of the test pulsing but also on the length of the wire being test pulsed. To make matters worse, in trying to predict the pulse height, there are gains on the pre-amps, shaper/amps and CCDs which can each give readings at plus or minus 10% of the actual value making the total error possible plus or minus 30%. The overall norm for the pulse height when test pulsing at the grid is between 50 and 400, although a smaller range can be established for each wire depending on its length. When test pulsing at the shaper/amp level, the acceptable range of values is from 190 to 400. Test pulsing the CCD is slightly different in that it is pulsed by a digital rather than analog signal, it is large enough to saturate the available bandwidth and gives readings of around 511. This means that the pulse height for the CCD cannot give a reading that is too high.

If the pulse height is abnormal in any way it could be indicating a bad board (which board depends on which sources that yielded good and erroneous readings); a missing or low pulse could indicate bad cabling or broken wires; and a high pulse indicates the shorting of two wires. A missing pulse is accompanied by a missing edge, missing pulse width and the number of pulses being equal to zero.

The Pulse Width is dependent on the pulser source and on the number of buckets in which the test pulse appears. For the grid its acceptable values are between 4 and 6; for the shaper the range is from 5 to 6 and the CCD's digital pulse is much wider with a width of 65 or 66. Problems shown in an abnormal pulse width when the source is the grid are with the pre-amp and when the source is the shaper/amp are with the

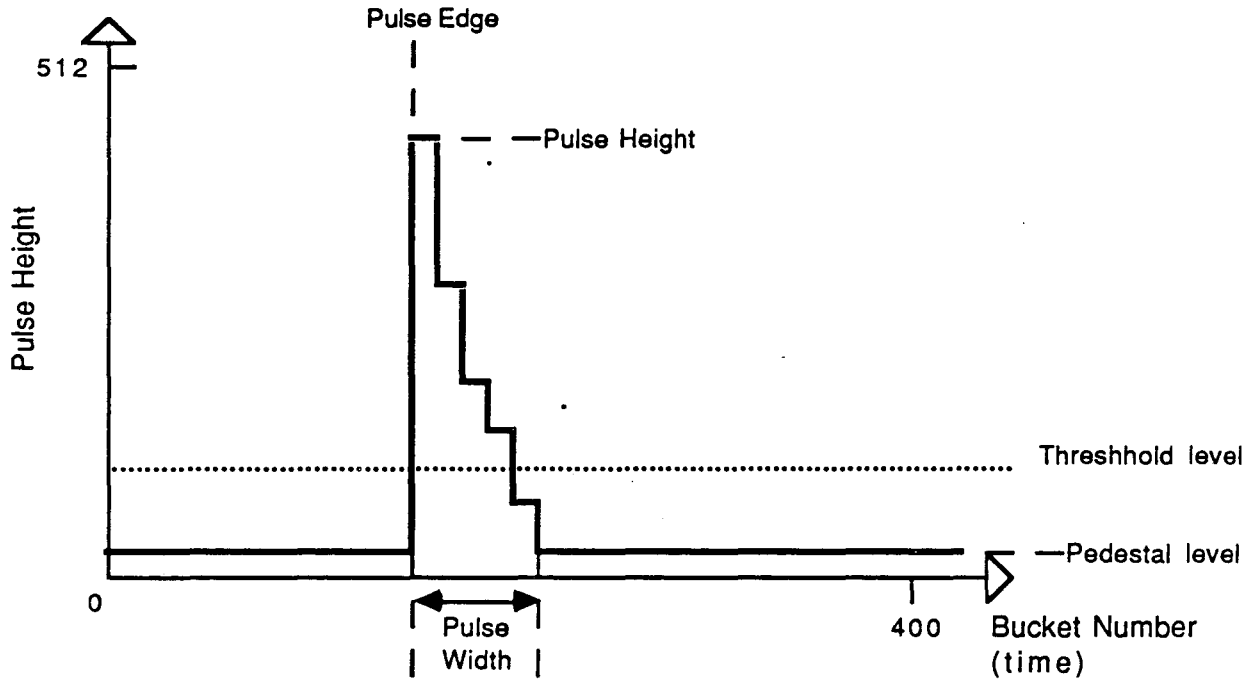


Figure 6.4a: **MonChn output of a normal channel test pulsed at the grid or shaper/amp level.**

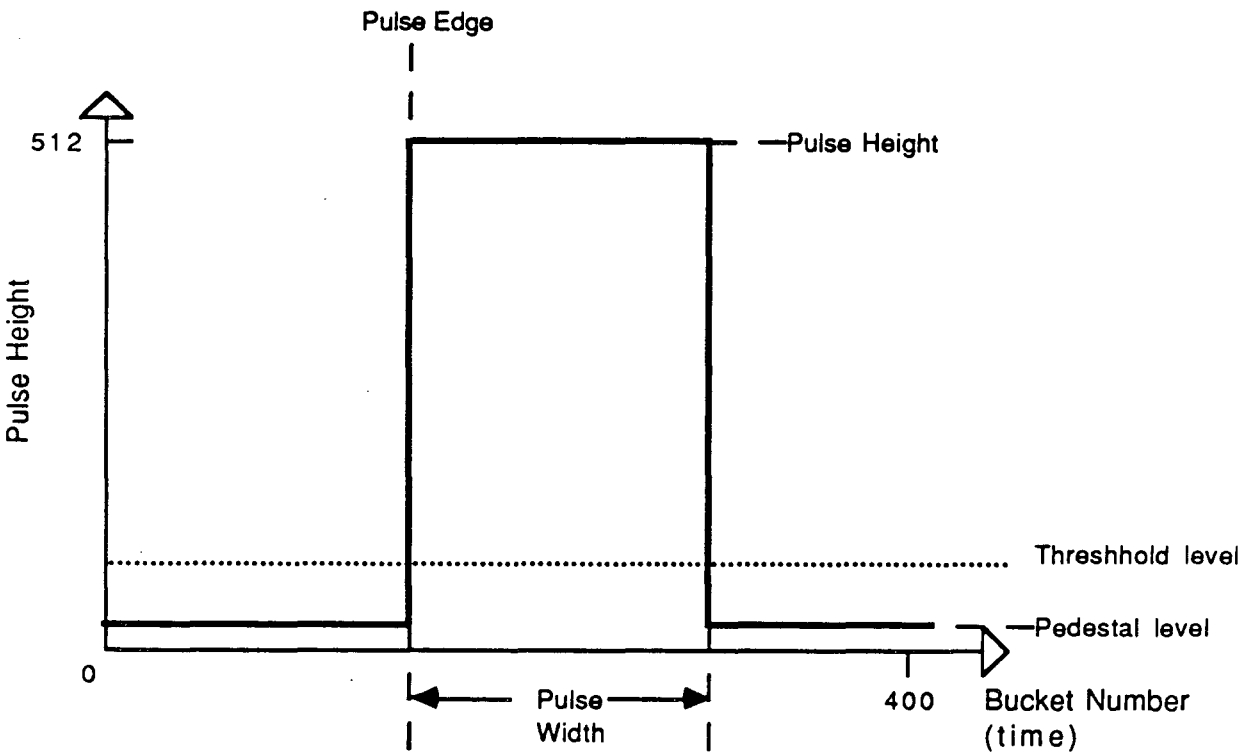


Figure 6.4b: **MonChn output of a normal channel test pulsed at the CCD level.**

shaper/amp.

The Number of Pulses are also given on the DeadCh program output, these are the number of (paired) crossings of the threshold level on the plot for a single input test pulse. The only acceptable value for this is 1, any other value denotes a failure of the channel. The two common failures diagnosed from an incorrect number of pulses are ringing and oscillating channels. Although a ringing channel has a normal edge and an oscillating channel will probably not, the two failures can be easily confused so the diagnostician must examine the plot given by MonChn to tell them apart. The differences in the plots of a ringing and an oscillating channel are shown in figure 6.4c and 6.4d. These problems are both caused by a faulty shaper/amp.

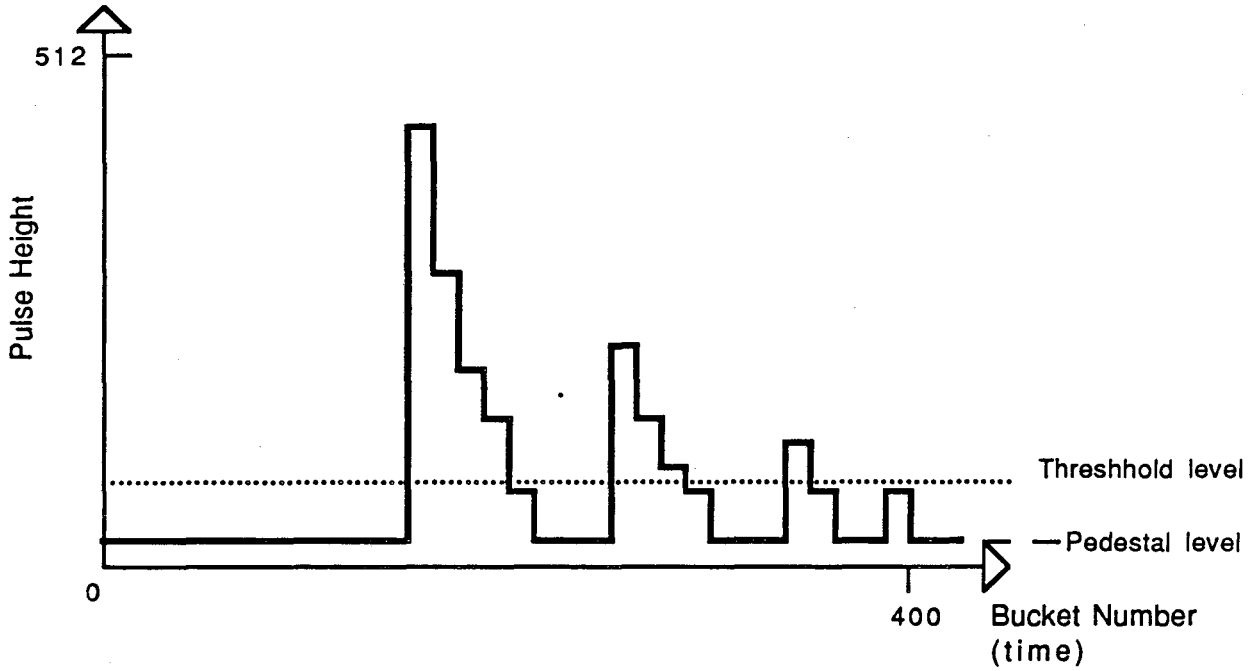


Figure 6.4c: **MonChn output of a ringing channel test pulsed at the grid or shaper/amp level.**

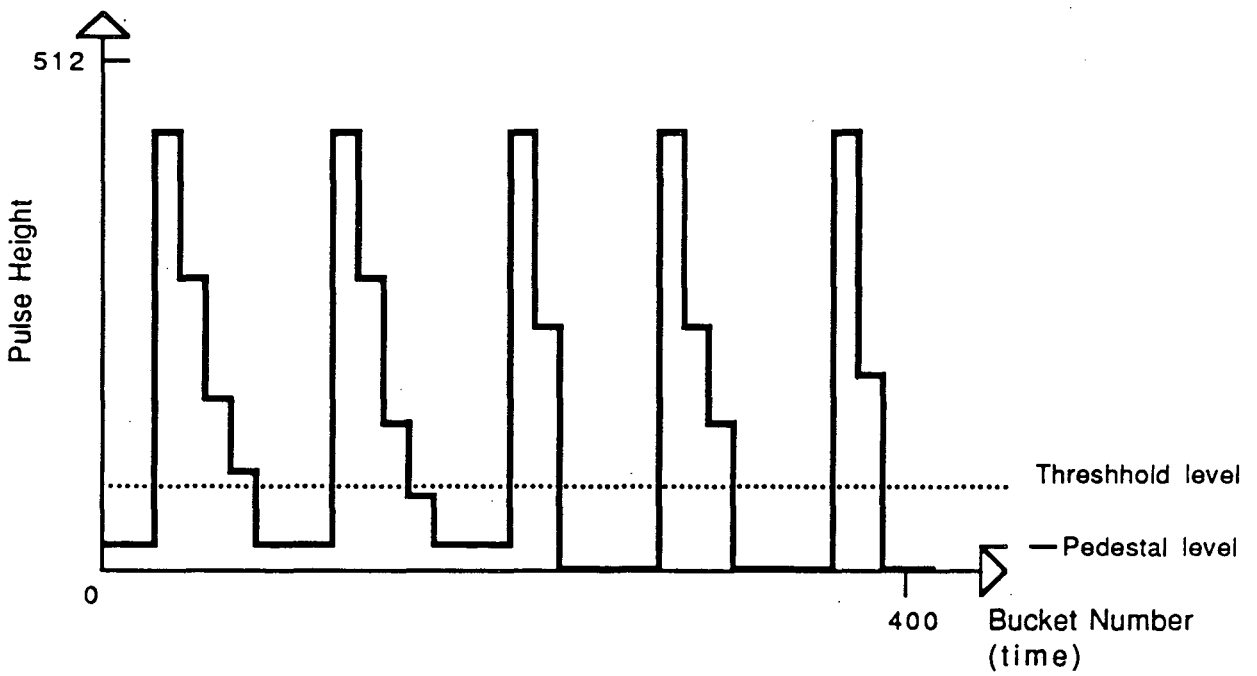


Figure 6.4d: **MonChn output of a oscillating channel test pulsed at the grid or shaper/amp level.**

7. Choosing an Expert System Shell for the TPC Application.

The expert system to trouble-shoot the hardware of the TPC was to be built using a commercially available expert system shell. Expert system shells have a ready built inference engine, methods of knowledge representation, usually the tools with which to build a user interface and some kind of developers interface. The developers task is to create a knowledge base by obtaining knowledge from the expert and encoding it into the shell using the tools provided. She must also interface the system with the outside world, creating a user interface and integrating the expert system with external programs, databases, etc.

To start with we established the shell requirements by considering the type of knowledge representation and control mechanisms necessary; the interfaces with hardware, software and the user and the cost of the shell. Commercially available expert system shells come with a wide variety of characteristics which range from the sophisticated, frame-based systems, running on large Lisp machines and costing in the region of fifty thousand dollars, down to simple, rule-based systems, running on PCs with a cost of just a few hundred dollars. Many shells lie between these extremes, offering object-oriented systems on a variety of machines and with price tags in the five thousand dollar range.

The next thing we did was to compare the various shells with these requirements and selected the one that we felt best fitted them. Finally we give a brief summary of Nexpert, the shell we selected, and of the software we intend to use with it, Ease+, to enhance some of Nexpert's weaker points.

7.1 Expert System Shell Requirements.

To establish which of the commercially available expert system shell best fitted the requirements of the TPC troubleshooting project, we must first ascertain what those requirements are.

Knowledge Representation.

The expert system shell must have frames or object-oriented knowledge representation to adequately show the relationship between different components in the TPC. This is also a more sophisticated method of knowledge representation than than a purely rule based system and therefore will be more useful in the future.

Secondly the shell must support as many rules as possible (in thousands rather than hundreds) since, although the initial prototype will only need somewhere in the region of a few hundred rules, future projects will add on to those substantially to make the complete TPC expert system.

The Control Mechanisms.

The control mechanisms in the shell must be a hybrid of forward and backward chaining. This is to allow incoming data to drive the reasoning processes (forward chaining) which is necessary if any monitoring capabilities are needed and allow the testing of hypothesis (backward chaining) to assess whether a certain failure has occurred.

It is also necessary to have both depth-first and breadth-first searching mechanisms. Depth-first to be able to fully explore a hypothesis, and breadth-first to assess what the possible hypotheses are.

The shell must be able to reason and search in the face of uncertainty in order to emulate real world problems where often, some of the information is not available. Finally the shell should be able to explain the route or actions it has taken in coming to it's conclusions.

The Interfaces.

The user interface should, ideally, be easy to create with facilities for color, menus and graphics to make the system as user-friendly as possible. This is important because many of the systems eventual users are either novices or infrequent users.

The developers interface must be easy-to-use, it must have a comprehensive and easy-to-learn rule entry language in order to speed up development time and make the transition between one developer and the next, as smooth as possible. There should be good training facilities for the developer, good vendor support and the expert system shell must have adequate debugging facilities.

The shell must be able to interface with other software packages, such as a databases, report generators, graphics, etc. It must also be able to pass multiple values both in and out of the expert system to (Fortran) routines and programs needed by the system to perform a variety of functions.

Finally, the interface with the hardware must be considered. The TPC detector is run exclusively on DEC computers (either PDPs or VAXs) and so the expert system shell must provide facilities for using the information from these machines and run on machines that are compatible with them. So ideally, the system would run on a VAX under VMS, but preferably could be developed on a personal computer or work-station for convenience.

The Cost.

The cost should be kept to the minimum necessary but incorporate as many of the above features as possible.

7.2 Appraisal of Commercially Available Expert System Shells.

Unsuitable Expert System Shells.

The following expert system shells were found to fall sufficiently short of the necessary requirements as to not be worthy of further investigation.

1ST CLASS is a very simple rule based system that does not support frame or object-oriented knowledge representation. It creates rules from examples. This expert system shell is not sophisticated enough for our purposes.

CxPERT fails on the requirement for a comprehensive and easy-to-learn rule entry language. It to make good use of this expert system shell one must first be proficient in C.

M.1. is an out of date expert system shell which although comparable in price to the suitable expert system shell mentioned later, it does not support frame or object-oriented knowledge representation, or many of the other features.

ART, KEE and S.1. are all very powerful, very large and very expensive expert system shells. They are worth considering only if nothing cheaper can perform nearly as well.

Expert System Shells That Meet the Necessary Requirements.

The following expert system shells meet most of the requirements. Here is an outline of their short-comings and their features which exceed requirements.

EXSYS 3.2 must be used with third party software, three different packages: EXCHANGE, FRAME & DBFRAME and TABLET, if it is to meet the required standards. This makes a comprehensive expert system shell fragmented, and possibly more difficult to use. EXSYS provides 5,000 rules, but it does not provide a hybrid control mechanism since it uses only backward chaining. It interfaces nicely with the user and developer, probably because of its relative simplicity, but it is difficult to tailor the interfaces to specific needs. The main advantage of EXSYS is that it is fairly cheap. Even including the third party software packages, the total cost for a PC is half of any of the other shells. However the VAX version is not significantly cheaper. The PC version of EXSYS is upwardly compatible with the VAX version.

GOLDWORKS short-comings are that it cannot reason with incomplete data, although it does use certainty factors. It can only call Fortran programs through C. The other major drawback is that it is only available on the IBM-PCs and not on the VAX or anything else.

PERSONAL CONSULTANT PLUS (PC+) is another promising expert system shell but is hard to evaluate positively as the demonstration package was for PC-easy, instead of PC+, and did not show some of the more sophisticated features. Technical help is very difficult to get hold of; there seems to be no established path. Apart from the poor support, PC+ is also limited in the number of rules it supports and in the hardware that it runs on, only the IBM-PC and various Texas Instruments machines. Again it provides no versions on the VAX.

7.3 The Chosen Expert System Shell - NEXPERT Object.

Although the above three expert system shells, Exsys, Goldworks and PC+, would all suffice for the project, it was felt that Exsys, having the price advantage, was much less sophisticated than the others and for this reason was rejected; Goldworks and PC+, both strong contenders and in a similar price range, finally lost to Neuron Data's NEXPERT Object. Despite Nexpert being probably the most difficult of the three finalists to learn, it is technically superior, being the most powerful and flexible. It was the one that fitted best the requirements of this project.

Knowledge Representation.

Knowledge is represented in Nexpert by using structured objects with a built in, user-defined inheritance mechanism. It is capable of both shallow or specific knowledge representation (the kind that is the easiest to obtain and is always used in knowledge processing systems) and deep knowledge representation or background knowledge (the type of knowledge that will make a system "expert" in its domain). Nexpert can use from 8,000 rules up, depending on the hardware it is running on, and an unlimited number of objects to represent the domain knowledge.

The Control Mechanisms.

Nexpert has a hybrid inferencing engine, being capable of both forward and backward chaining. This can be user-defined globally by the rules or locally for a sub-set of the rules. The method of searching, again user-defined, can be done depth- or breadth-first. During an inferencing or knowledge processing session reasoning can be event-driven; truth is maintained and objects can be created or removed from the knowledge base as they are needed.

Nexpert is not well equipped to deal with uncertainty, it only offers the possibilities of a hypothesis being true, false, not-known or unknown (the difference between the last two possibilities is that not-known is the case in which the hypothesis has been tested and the answer is not known and unknown is the case in which the hypothesis has not been tested). Nexpert offers no measures of uncertainty.

Nexpert keeps a log of the route taken during an inferencing session which can be analyzed later by the user/developer, to show how the inferencing was done.

The Interfaces.

The User Interface tools of Nexpert are primative, offering prompt lines for data input by the user but little use of graphics (except for integrating with other software packages such as DrHalo for static images) and limited or no use of menus and dynamic images.

The Developers Interface tools are run under a windowing environment and offers editors for entering rules, the strategies for inferencing, objects, their classes, properties and metaslots. Rules are compiled incrementally, making it easy to run the inference engine at any time during the actual encoding of the rules but it also has the drawback of making the rule editor annoyingly slow to use. For debugging, there are windows that will show the conclusions as they are reached, the data as it is instantiated and the route the inferencing engine took, as well as graphically displayed networks that show how the rules and objects are structured.

Bechtel AI Institute, located in San Francisco, provide the user support, including an electronic bulletin board, telephone help and training classes.

The Software Interface is adequate. Nexpert can use a variety of database packages and run any executable (compiled) program from within it. Passing variables in and out is a little more tricky, but still possible.

The Hardware Interface is the most comprehensive of the expert system shells. Nexpert can run on the VAX machines, the IBM-AT, the Mac II and under UNIX. All the versions are compatible with one another, so it is possible to develop the expert system on a PC or work station and then use the runtime version on the VAX, an ideal setup for this project.

The Cost.

Nexpert, at $5,000 to $8,000 falls in the middle of the price range for expert system shells which go from a couple of hundred dollars (1st Class) to tens of thousands (ART - $65,000).

Choice of hardware.

Once we had decided to use NEXPERT as our expert system shell, we needed to decide on which machine we wished to develop the system. For the development stage of the project we cut the choice down to using either the Mac II or the IBM-AT as these machines were available. The Mac II version of NEXPERT has only just been released, it is faster than the IBM-AT version and supports an unlimited number of rules. Its disadvantage is that it only supports one database package which is dBase III+; the dBase III+ implementation on the Mac has been very poorly reviewed. It has evidently lost a lot of its power and is much more difficult to use. So the IBM-AT, while being slower and limited in rules (to 8,000) is backed up by much better software. DBase III+ works very well on the IBM-AT. In addition there is a very useful software package which is designed to work with NEXPERT on the IBM-AT and later on the VAX, called EASE+ (description below). The choice of machines comes down to the choice between superior hardware on the Mac II and superior software on the IBM-AT.

The IBM-AT was chosen for the development machine for this project for three reasons. The main reason is that we are able to use the application development environment, EASE+, which will upgrade the poorer aspects of Nexpert, to make the creation of a comprehensive expert system application much easier. Secondly, it is very useful to have a good database such as dBase III+ to use for the storage of the static data, easily accessible through Nexpert's database interface. The third reason is that the expert already has an IBM-AT which he uses constantly, and so may feel more comfortable with it and it will possibly keep the costs for the later stages of development down.

7.4 EASE+ - an application development environment.

EASE+ is a package which is used to integrate the software of an application, and provide a good user interface with dynamic graphics, menus and lists. It also has an object-oriented database which is used to store the information needed to support these features, but which can also be utilized by the developer for any other purposes. EASE+ has been combined with Nexpert by its developers, Expert-EASE Systems Inc., to make the two packages easy for expert system developers to use, and it incorporates at least three major features to enhance Nexpert.

The first of these is to provide a method by which Nexpert can reason with uncertainty, using the Dempster Shafer Theory. This allows one to assign measures of belief, possibility and ignorance to data, rules and facts. It will use these values to calculate the likelyhood that the conclusions it reaches are true.

The second of the enhancements is the ability for EASE+ to pre-focus the knowledge base, a feature that is particularly useful for developing a system that must cater for a range of user, from the novice to the expert. This feature allows the expert user to specify the area he believes the failure has occurred and the knowledge processing will, at least initially, confine its search to that area, speeding up a consultation, when normally the system is designed to nurse a novice user.

The third enhancement makes the integration of other pieces of software with Nexpert easier to handle, with simplification of the data transfer between Nexpert and other software. In addition EASE+ can handle the interruption and resuming of a Nexpert knowledge processing session.

Thus EASE+Nexpert, make it possible to build a complete expert system which include all the previously mentioned requirements for an expert system and for this project.

8. Building the Knowledge Base for the Prototype Expert System.

Once the preparation phase has been completed, a basic understanding of the problems involved in troubleshooting the electronics of the TPC had been gained by the knowledge engineer and the expert system shell has been chosen, the building of the prototype expert system could begin. The prototype was to be comprised of a number of knowledge and data bases, each of which had to be designed.

The Data and Knowledge Bases.

The knowledge representation task was split into four main sections, the static data, the static knowledge, the dynamic data and the pattern-matching and experiential knowledge. The static data is used to follow a channel to its different location addresses and find the equipment used to test pulse any particular channel. This information is encoded in the databases. The static knowledge encompasses the deeper knowledge of the system in a model of a channel, encoding structural and functional information of a channel in a Nexpert knowledge base. Dynamic data, or the input data, is the data that is used to analyze the TPC for failures. Finally, the compiled, pattern matching and experiential knowledge is the expertise used by the expert in coming to his solutions.

To describe the knowledge bases diagrams have been used to show the relationships between objects and their classes. Triangles have been used to represent objects and circles to represent classes. Solid symbols represent the permanent objects or classes, which are created while the system is being developed, and hollow symbols are used for dynamic objects and classes that are created during a knowledge base consultation. Dynamic objects are named (unless explicitly told otherwise) using the class name of the permanent class to which it belongs concatenated with a variable of some sort to distinguish it from the others in its class.

8.1 Static Data.

Static data consists of the data used to describe the addressing scheme of the TPC. This information has been encoded into a relational database using dBase III+. Database and programming files in dBase have been written so that it is possible to get from any of the three different addresses to any of the others easily. It is also possible to find the address of the test pulsing equipment for any given channel, but searching for a channel given the test pulsing equipment is not implemented because there is no need to since one always specifies the channel.

Presently, the information for all the above is embedded in Fortran subroutines. The aim of producing a database for this information rather than using the subroutines, was to provide faster, easier method of access for Nexpert. However, this part of the system was written before we had received a copy of Nexpert or Ease+ and was written on the assumption that Nexpert interfaces with dBase III+. It was found out later that although there is a database file interface, the dBase programs which link the relational database files together for the complete database could not be executed directly from Nexpert.

8.2 Static Knowledge.

The static knowledge consists of the information that represents the configuration of a channel within the TPC. It is knowledge that is unchanging, it remains the same for each consultation being independent of the input data. The path of the TPC channel has been written into a Nexpert knowledge base. All the components along the path have been made into objects belonging to the class of components within the knowledge base (figure 8.2a), in addition, each of the components belong to a single location class (figure 8.2b) and one or more areas of influence classes (e.g. test pulsing area, trigger system area or readout area as shown in figure 8.2c). Each of the areas of influence encompasses a number of the components that the expert looks at when a failure is thought to be caused by that influencing factor. For example, a failure in the readout will cause the expert to look at the CCDs, digitizers, readout master controllers and line drivers of the affected channels. The components or objects all have a number of properties associated with them whose values are unique, for example, their function, and position in the path. In addition, these objects have properties whose values are inherited from their location class (e.g. the rack in which they are situated relative to the digitizers) and area of influence classes (e.g. the type of output signal they generate). All of these properties have values permanently assigned, making this knowledge static in nature.

8.3 Dynamic Data (Input).

Dynamic data consists of the data or results to be analyzed. The output from the test pulsing program, DeadCh.ftn, which is to be used as input into the troubleshooting system, is an ascii file containing the full address and results of the test pulsing for each of the channels in the test set. The results of the test pulsing are the five numerical values accompanying the channel's addresses and are the pedestal value, the edge, height and width of the output pulse along with the number of pulses detected.

This information is used for troubleshooting and so must be input into the system. Three different methods, described below, were tried for representing this data. As we learned more about the characteristics of Nexpert and the representations needed for the troubleshooting, successive versions of the dynamic data's knowledge base were designed. Here, is a description of each of the three versions with the reasons for abandoning each method in favor of its successor.

Version 1.

For each channel under investigation, an object is created within a Nexpert knowledge base. Then sub-objects are created to contain the raw numerical data (read in via the database interface of Nexpert) and a second to contain the processed data or results (with values of high, normal, ...). These sub-objects are named according to the position at which the channel was test pulsed e.g. the grid (GRD), the shaper/amplifier (SHA) or the CCD. So a channel could have up to six sub-objects attached to it. Rules are used to convert the raw numerical data into quantized values. This part of the KB has been written and using simulated (not real) raw data, it works. Figure 8.31 shows how the objects and classes are related in this KB.

Problems.

There are two major problems with this method of input data representation. Firstly, the channels in this KB are not structured in any way and so it would be difficult, say, to establish which channels are on the same board or in the same list or bin. This kind of information is needed when analyzing the results and must be incorporated at this stage. Secondly, using the knowledge base to convert the raw data into results was found to be very inefficient both in development and running time because this is a largely procedural process that would be more efficiently done using an external program or procedure.

Figures 8.2: The Path of a Channel

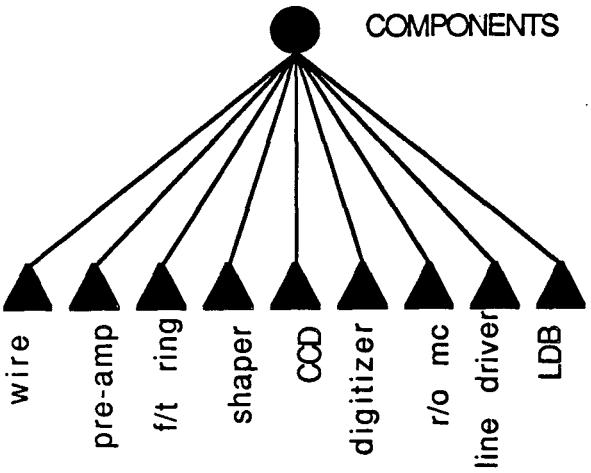


Figure 8.2a

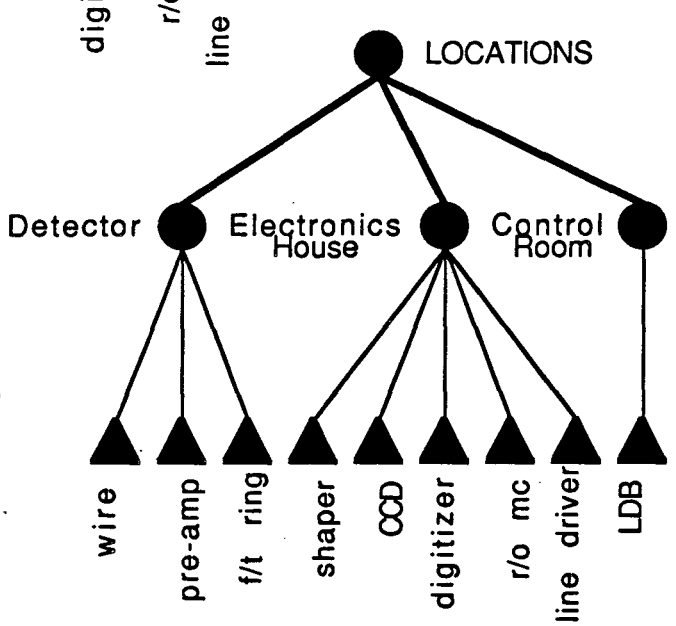


Figure 8.2b

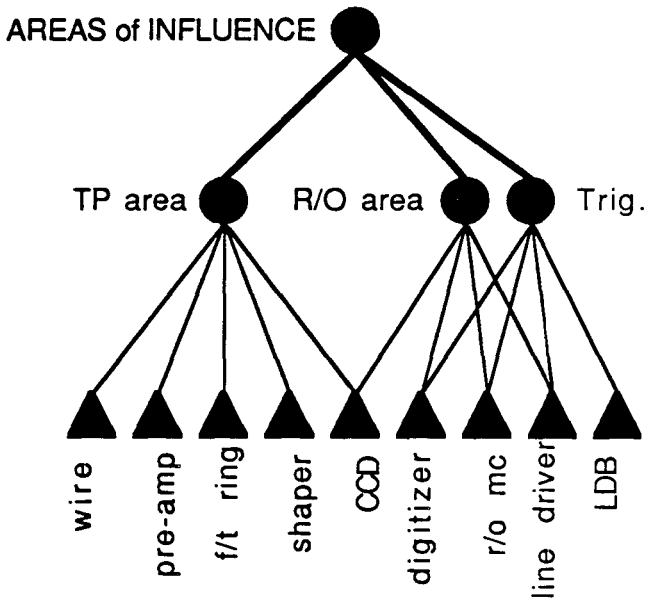


Figure 8.2c

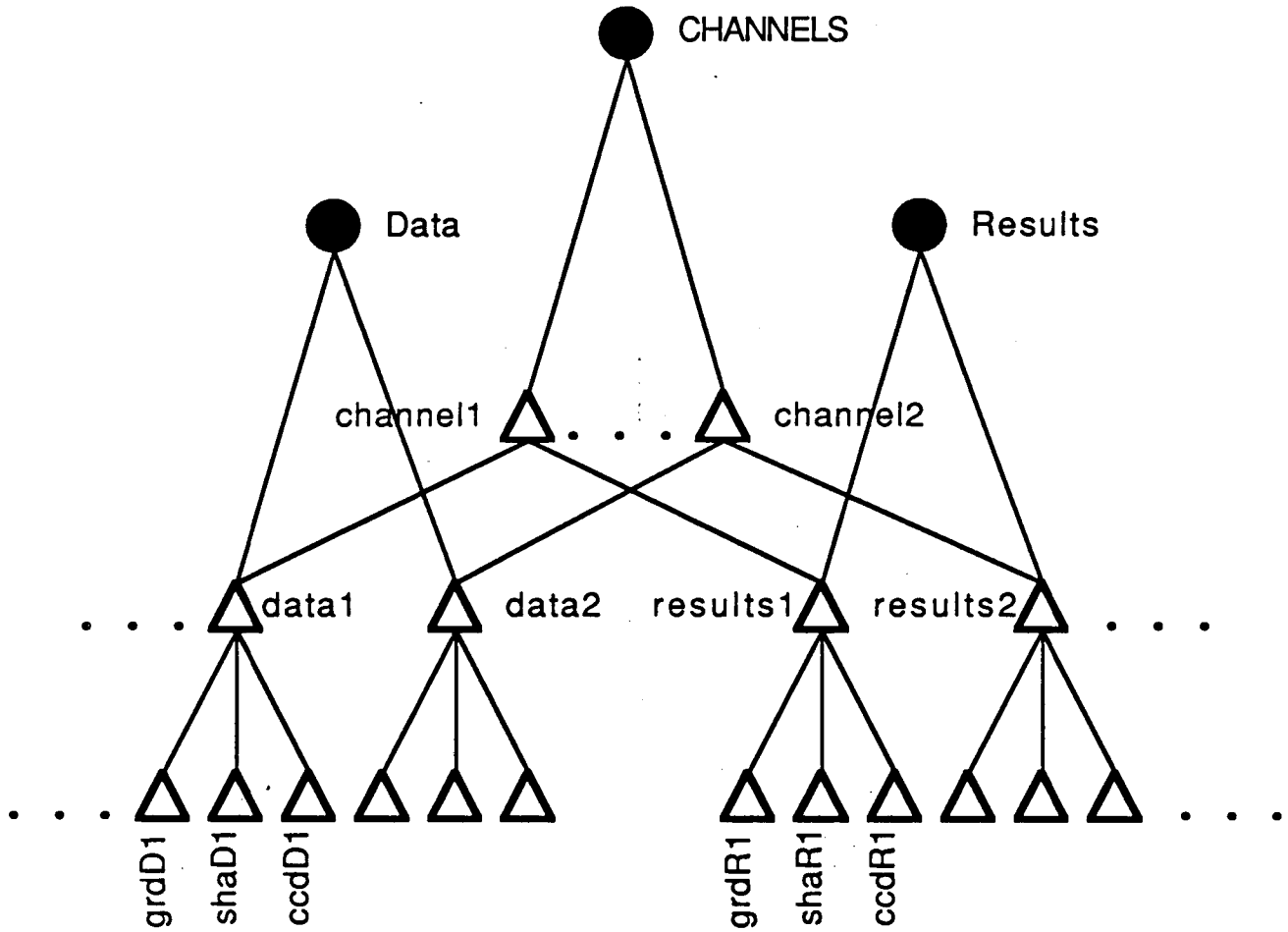


Figure 8.31: **First attempt at data input (Version1.kb).**

Version 2.

For this version of the input data representation we have assumed that the data has been pre-processed so that the test pulsing results are qualitative, not numerical, with values of high, normal, low or missing. The channels and their associated results are created, as objects and sub-objects in a structured manner.

At the start of a consultation with the knowledge base StruChn, the address of the first channel of the test set is passed from a dBase III+ file into the Nexpert KB. The user is asked whether the test pulsing has been done by list or by sector. The six wire lists and six sectors of an end-cap are represented as static objects (figure 8.32a). Each of the twelve static objects have static values assigned to their properties, for example, a list object, a member of the class LISTS, has the properties endcap, row, rack and bin, which contain its unique set of static values. So once the correct list or sector has been selected, using the information from the input data and the responses to the questions, the KB creates all the groups (or boards) to be associated with the chosen list or sector (figure 8.32b). These groups are objects created within the class GROUPS and sub-objects within the selected list or sector; they are created from the fields of the appropriate database file, for example, the groups for list5 will be created from byList5.dbf. The database files contain the property values for each group (board) in a particular list or sector.

Next, the group is selected, using the input address, and the channels belonging to that group are created as objects in the class CHANNELS and sub-objects of the group object (figure 8.32c). Sixteen channels are created in a group; they inherit their properties from CHANNELS and the property values from their group.

The final stage of the data input in StruChn.kb is the creation of results objects. A channel is selected, again, from the input address and the user is asked whether the test pulsing took place at the grid, shaper/amplifier or CCD positions. Based on this information and a knowledge of the history of this consultation (i.e. how many times the test pulsing has been repeated) a results object is created in the appropriate sub-class of DeadChRSLTS (DeadCh results) and as a sub-object of the selected channel. In addition to the ability to create results objects for each of the different test pulsing positions, the KB can also cope with up to two repeats at that same position, making it possible for each channel to have nine results objects associated with it (figure 8.32d). The results objects have their property values read in from a dBase III+ file containing the pre-processed results.

This whole procedure is repeated, at each level checking to see if the required objects already exist before creating new ones, until there is no more input data in the address of results database files. For an input file containing only one group of objects with two repeats of one of the channels results and all test pulsing done at the grid position the resulting structure would look like that in figure 8.32e.

Problems.

The problems with this version arose when the diagnosis knowledge started to be added, although it was possible using a number of tricks, duplicate rules etc., to overcome many of these the KB quickly became too complex to be easily understood which, after all, is one of the reasons to use this technology, rather than just another fortran program.

Nexpert uses a construct it calls pattern-matching which makes it possible to ask questions like "Do any of the objects in the class CHANNELS have the property groupNo with a value of 5?" within a rule. This construct creates a list, local to that rule, of all the objects that satisfy that condition. Such a statement is found on the left-hand side of the rule and only allows the rule to fire if there is at least one object within the list. This pattern matching construct is essential in the diagnostic section of the KB. A problem with this method of structuring the input data arose when

Figures 8.32: **Second attempt at data input (Version2.kb).**

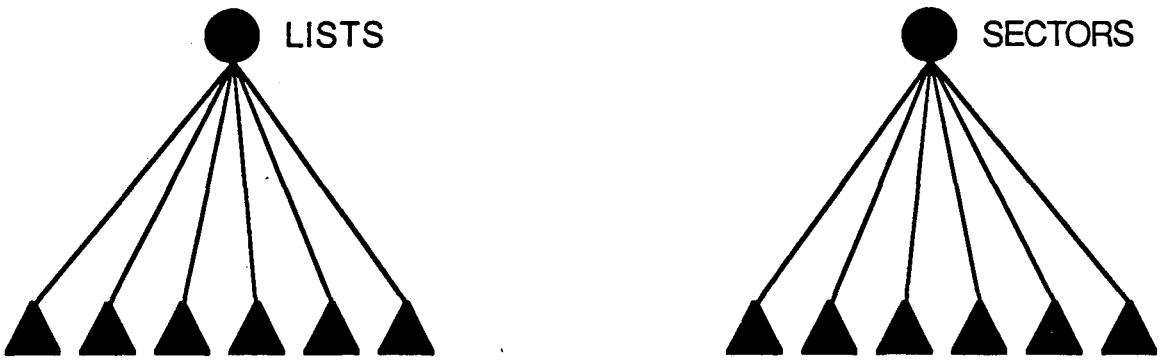


Figure 8.32a

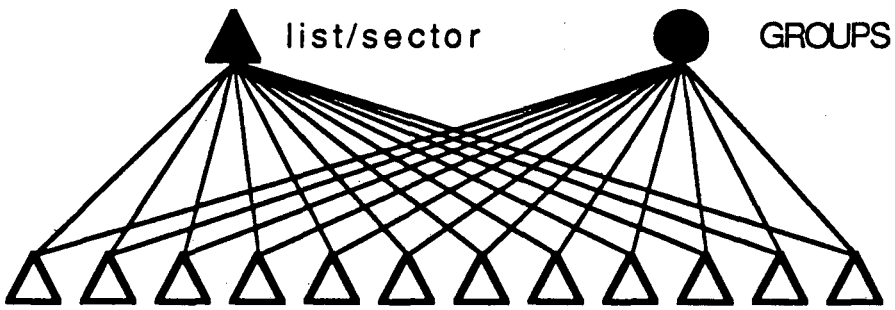


Figure 8.32b

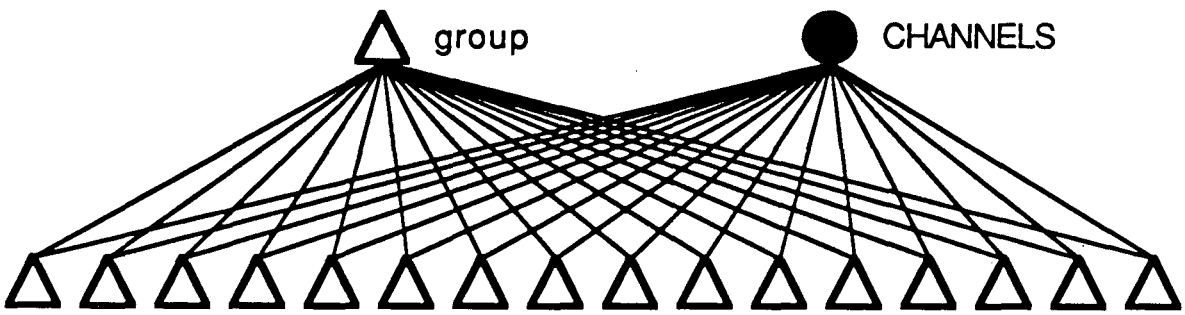


Figure 8.32c

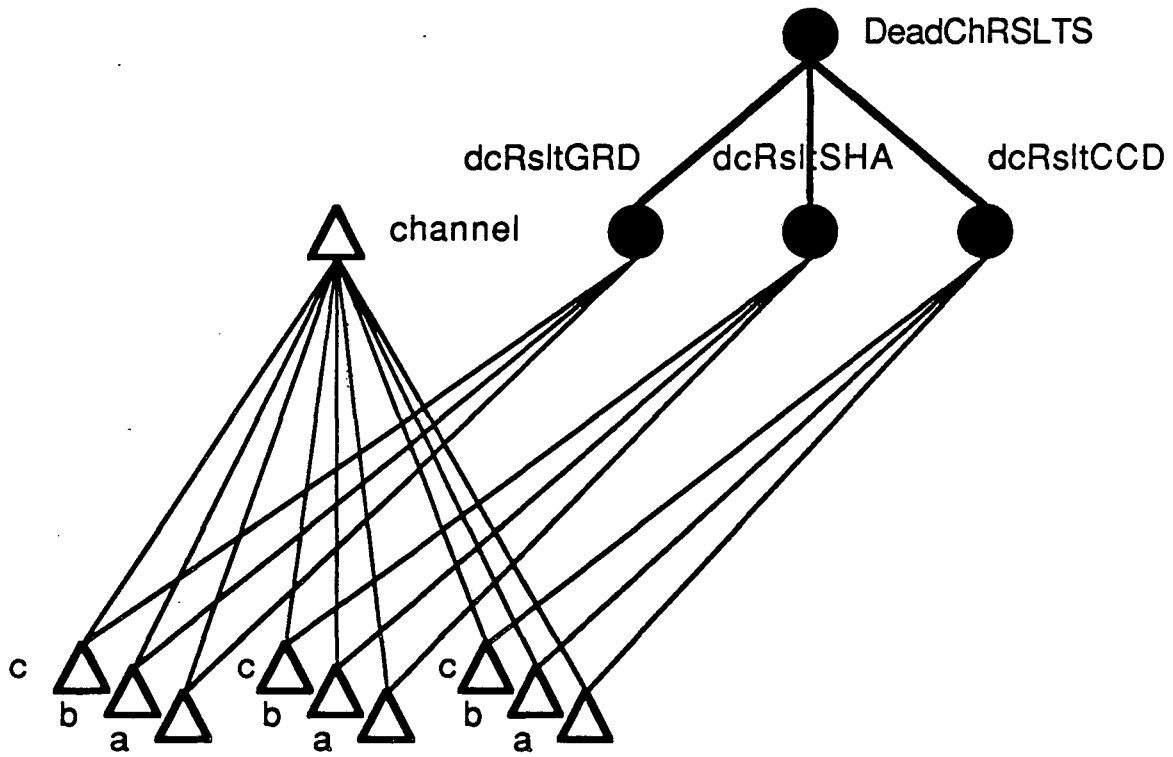

DeadChRSLTS
dcRsltGRD
dcRsltSHA
dcRsltCCD
channel
c
b
a
c
b
a
c
b
a

Figure 8.32d

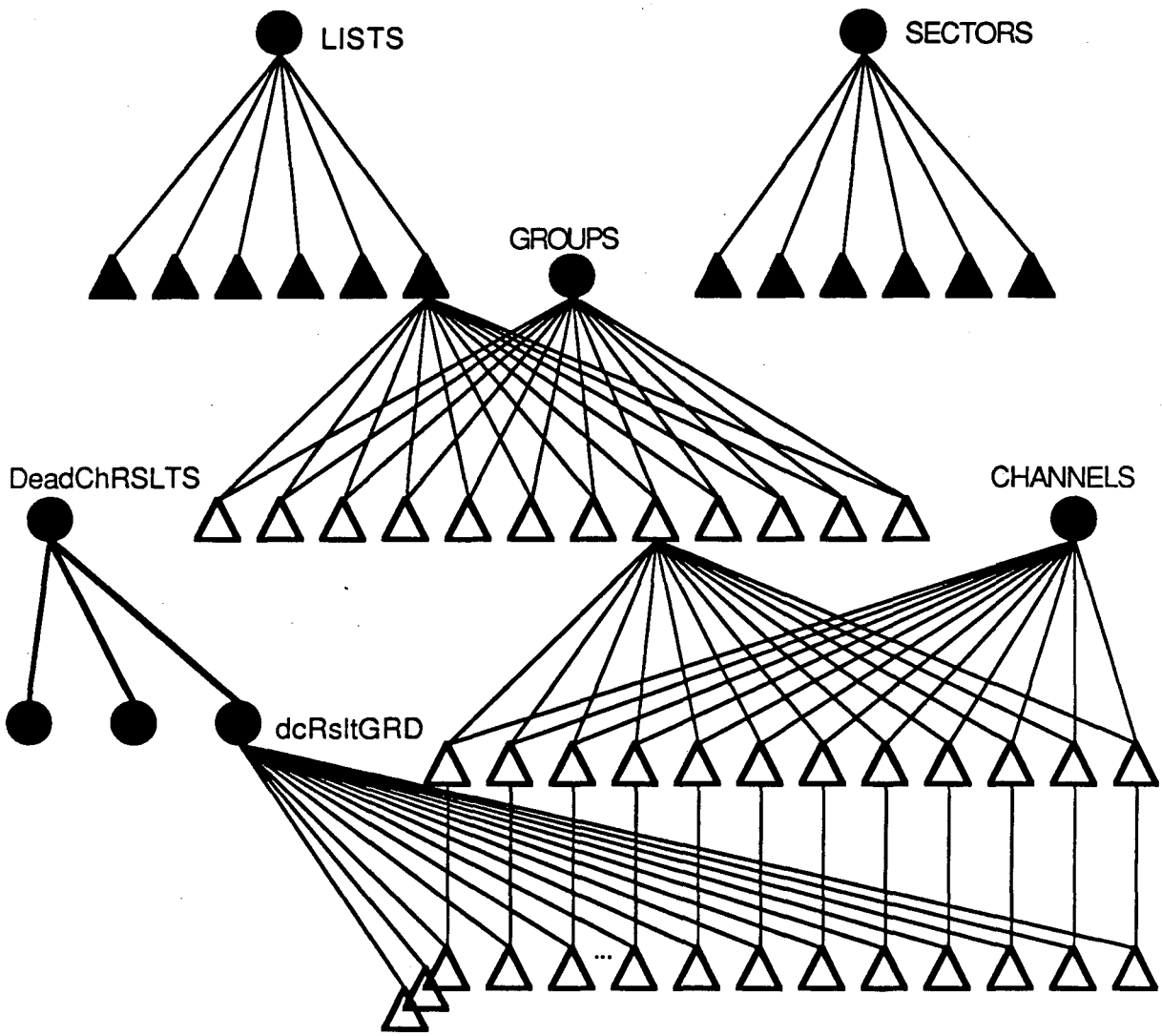


Figure 8.32e: **An overview**

limitations of the pattern-matching construct were found.

Dynamic objects are created in this KB by concatenating a root with a variable ("root"\variable\) in what Nexpert calls interpretations, an example of this is "channel"\chanNo\ which will produce object names such as channel1, channel362 and so on. Each object must have a unique name. It is not possible in the current version of Nexpert (it has been promised for the next version) to use these interpretations within the pattern-matching construct, so we cannot ask "Do any of the objects within "channel"\chanNo\ have the property groupNo with a value of 5?". It is impractical to create rules to ask this same question for all the possible instances of the channel objects. A way to get around this is to create, in a previous rule, a temporary class containing the objects one wishes to use in the pattern-matching, but this involves all the objects of the super-object having at least one common property value to each other and different from all the other objects in their class so that pattern-matching can be used on the entire class of objects to create the sub-class that matches the original super-object.

A second problem with this structured approach to data input and pattern-matching is that one cannot compare the properties of sub-objects of the objects within a class, for example, you cannot ask "Do any of the dcRsltGRD sub-objects of the channel objects of the class CHANNELS_UNDER_INSPECTION have a high pedestal property value?". So again the relevant sub-objects must be put into a separate class temporarily.

Finally, we found that there is no mechanism in Nexpert to directly tell you which classes and super-objects a particular object belongs to. This can be done indirectly by having a property set up in the object for each of the possible classes and super-objects that it may belong to and have a value, say the class name, inherited down to the object when it is assigned to that class. For an example of let us say that group57 is created within list5, both the group and the list must have the same property such as listNo. ListNo has the value 5 already in it and this value is then inherited down to group57's listNo property.

All of these problems make the structuring of the input data an unnecessary complication, especially as the major reasons for doing this were to cut down the amount of repeated information and simplify the rules in the diagnostic section of the knowledge base. Neither of these were achieved using this structure due to the inherent limitations of Nexpert version 1.0.

Version 3.

This is the final and by far the simplest version, assumes some pre-processing of the results making their values qualitative as in version 2 and using most of the addressing information which accompanies a channels results in the DeadCh.ftn programs output.

In this version one object is created for each line of the output in other words for each channel within the CHANNELS class (figure 8.33). The properties of these channels include all the addressing information as well as the results. At present there is no allowance made for storing repeat results or results from test pulsing at the different positions. All the channels present on the DeadCh output are created at the beginning of the session.

This is the method of data input used at present, on which the diagnostic rules have been added.

Problems.

The main disadvantage with this version is that it creates a large number of objects which significantly slows down the consultation with Nexpert's knowledge base. This is due to the inefficient methods of memory management Nexpert uses which Neuron Data

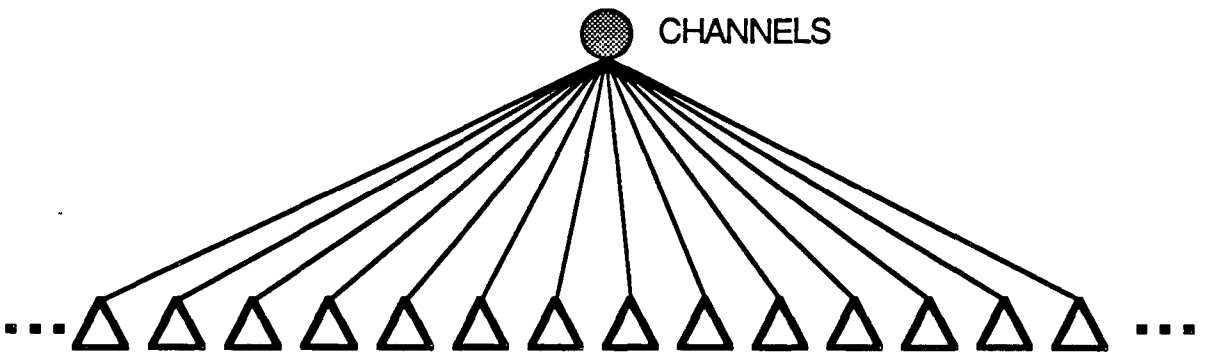


Figure 8.33: **Third and final attempt at data input.**

have promised to improve on in the next version of Nexpert (version 1.1). To overcome this problem we considered further preprocessing the results of channels and only inputing those which were abnormal. This would significantly reduce the number of objects being created in a session and so speed up the knowledge processing.

8.4 Compiled Pattern-Matching and Experiential Knowledge.

Once a method for inputing the data had been devised, we turned our attention, in the final few weeks of the project, to using that data for troubleshooting the electronics of the TPC. The data, as we explained in the previous section, was input as objects which are called channels. This stage of the knowledge base development can be split into two, the pattern-matching or classification step and the experiential knowledge encoding step. The classification of the channels was done first according to their results and then, from within those initial classes, they were further classified according to whether the channels in a particular class are on the same board in the electronics house or not. The second stage, encoding the experiential knowledge (or the troubleshooting rules), was not reached in time and so has barely begun. However some thought has gone into the type and nature of the rules which need to be encoded. This is discussed in the final section of this part of the paper.

Developing the Pattern-Matching Knowledge Base.

Before the trouble-shooting rules were started, we decided to sort the channels by assigning them to classes according to the values of their result properties. We believed this would simplify the writing of the troubleshooting rules when we got to them.

The first step is to sort the channels into those that are working normally and those that have failed in any way. This is done using the pattern matching construct of Nexpert and creating a list of all the channels within the class CHANNELS whose result values are all OK and then linking them to the ChannelOK class (figure 8.41a). The remaining channels are put within the Failure class (figure 8.41b) before being further sub-divided.

The second step is to sort the channels belonging to the Failure class. Again using the pattern-matching construct, a list is created of all the channels within the class Failure whose pedestal value is not normal, these channels are linked to the class PedestalFail and their link to the Failure class is removed (figure 8.42a). The class PedestalFail is a sub-class of Failure so the channels are still within the Failure super-class. The remaining channels directly linked to the class Failure are re-linked into the class PulseFail (figure 8.42b). The assumption is that if a channel is not working correctly and its pedestal is OK then there must be something wrong with one or more of the other results values (the pulse height, the pulse width, the pulse edge or the number of pulses), all of which indicate a problem with the pulse.

The third step, then, is to classify the channels within the PulseFail class. These are sorted in a very similar way to one above, this time creating a list of all the channels within the class PulseFail whose pulse height, width, edge or count values are missing and moving them from PulseFail to its sub-class MissingPls (figure 8.43a). The remainder of the channels in PulseFail are put into the class AbnormPls (figure 8.43b). The channels in the class AbnormPls all have an abnormal pulse, with at least one of the values pulse edge, height, width or count being either high or low but none of the values being missing.

The channels are now classified according to the values of their result properties as shown in figure 8.44, whilst still belonging to the CHANNELS class.

Figures 8.41: **Step 1 in classifying the channels according to their results**

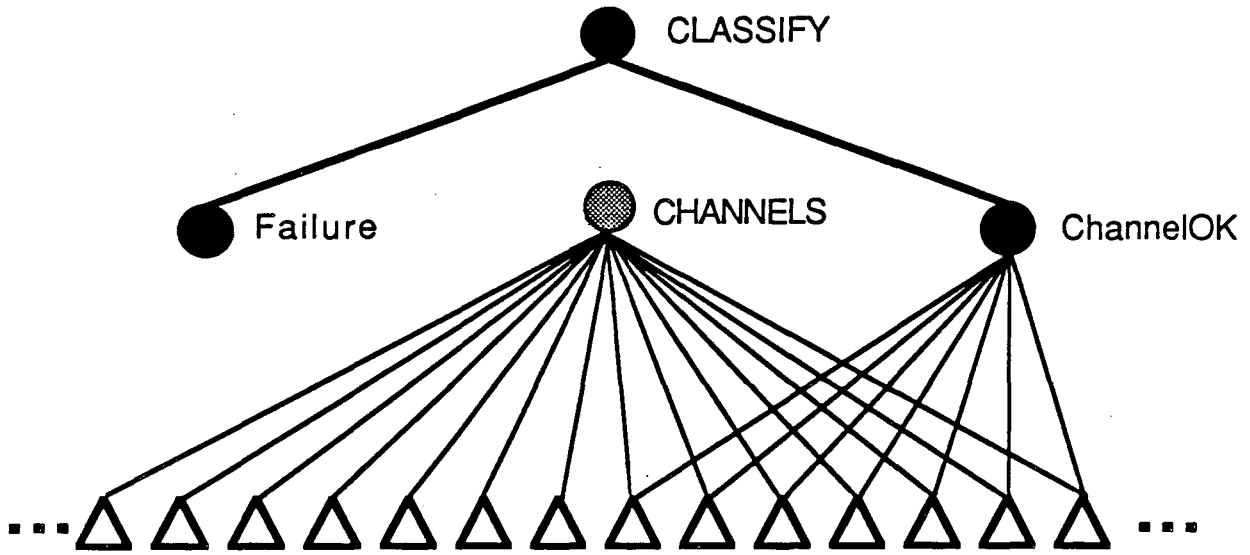


Figure 8.41a

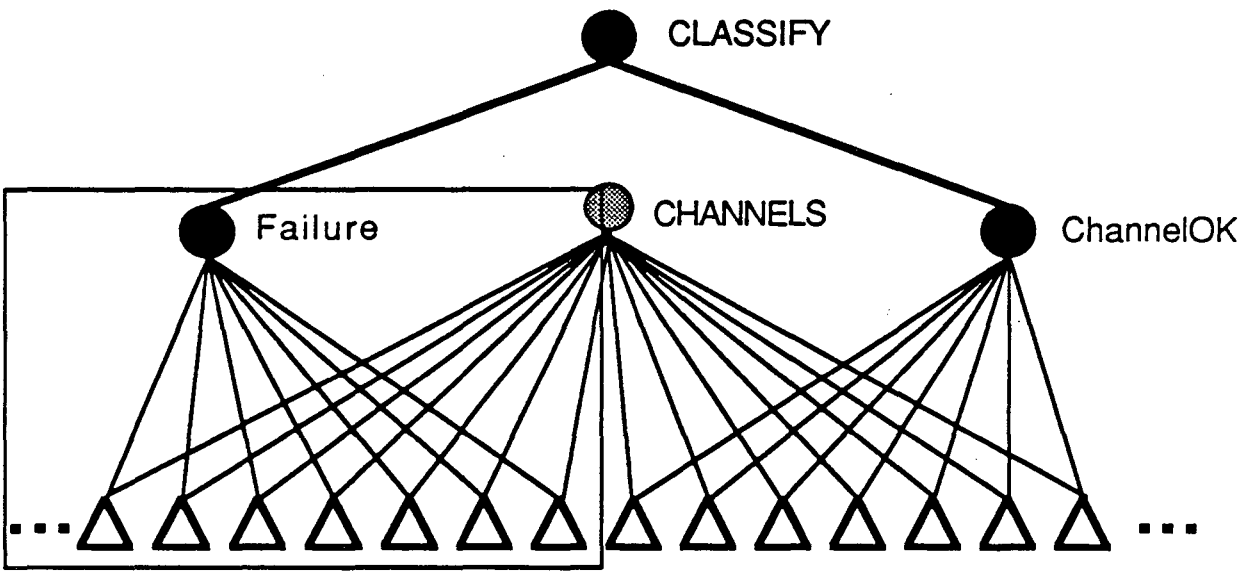


Figure 8.41b

Figures 8.42: **Step 2 in classifying the channels according to their results**

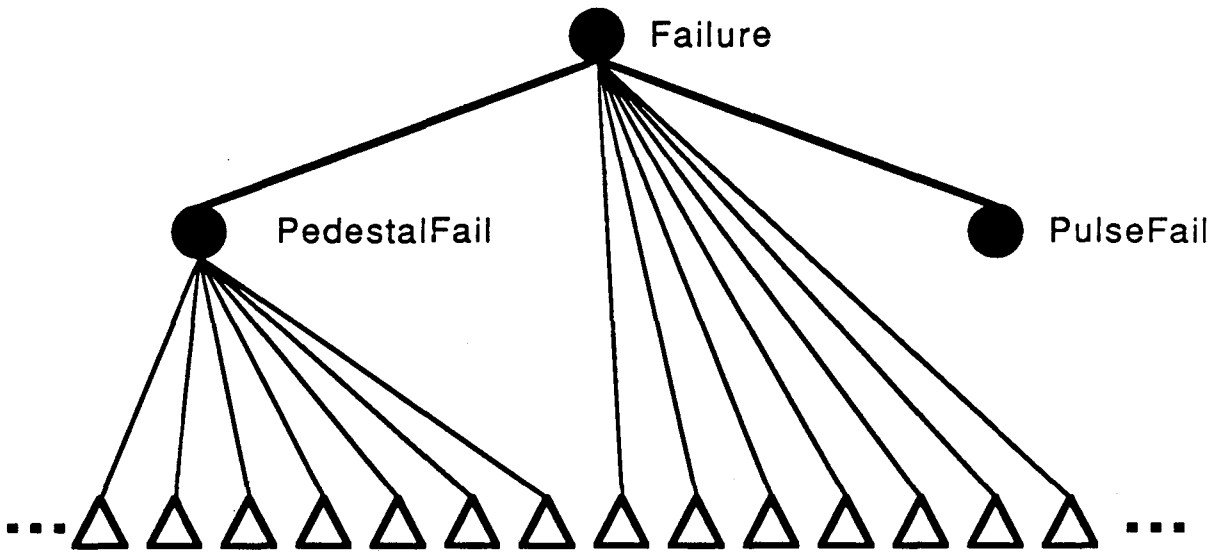


Figure 8.42a

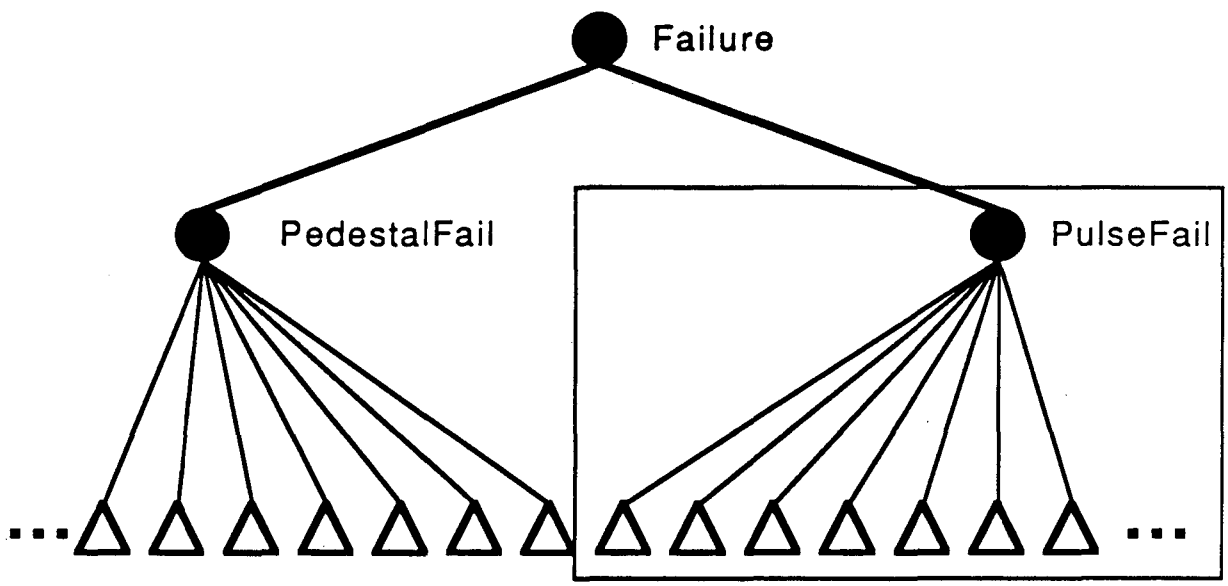


Figure 8.42b

Figures 8.43: **Step 3 in classifying the channels according to their results**

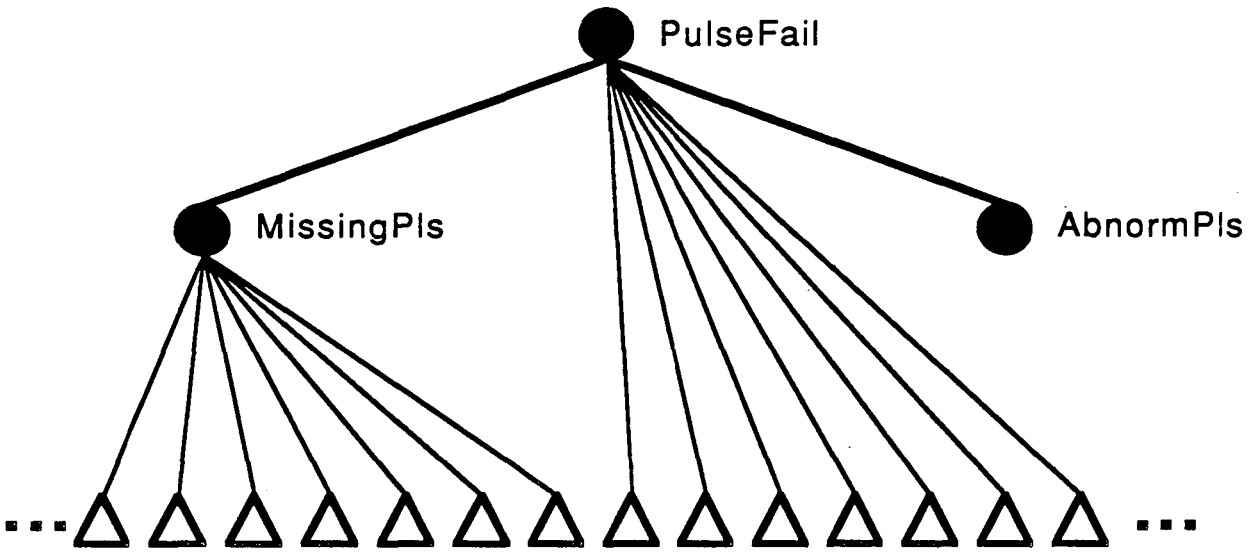


Figure 8.43a

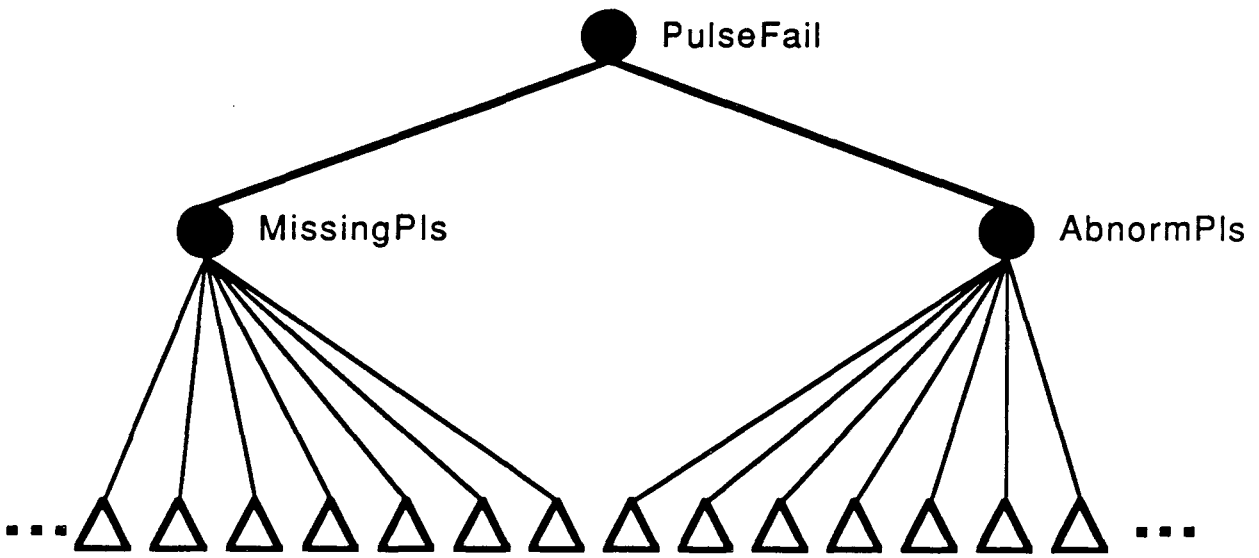


Figure 8.43b

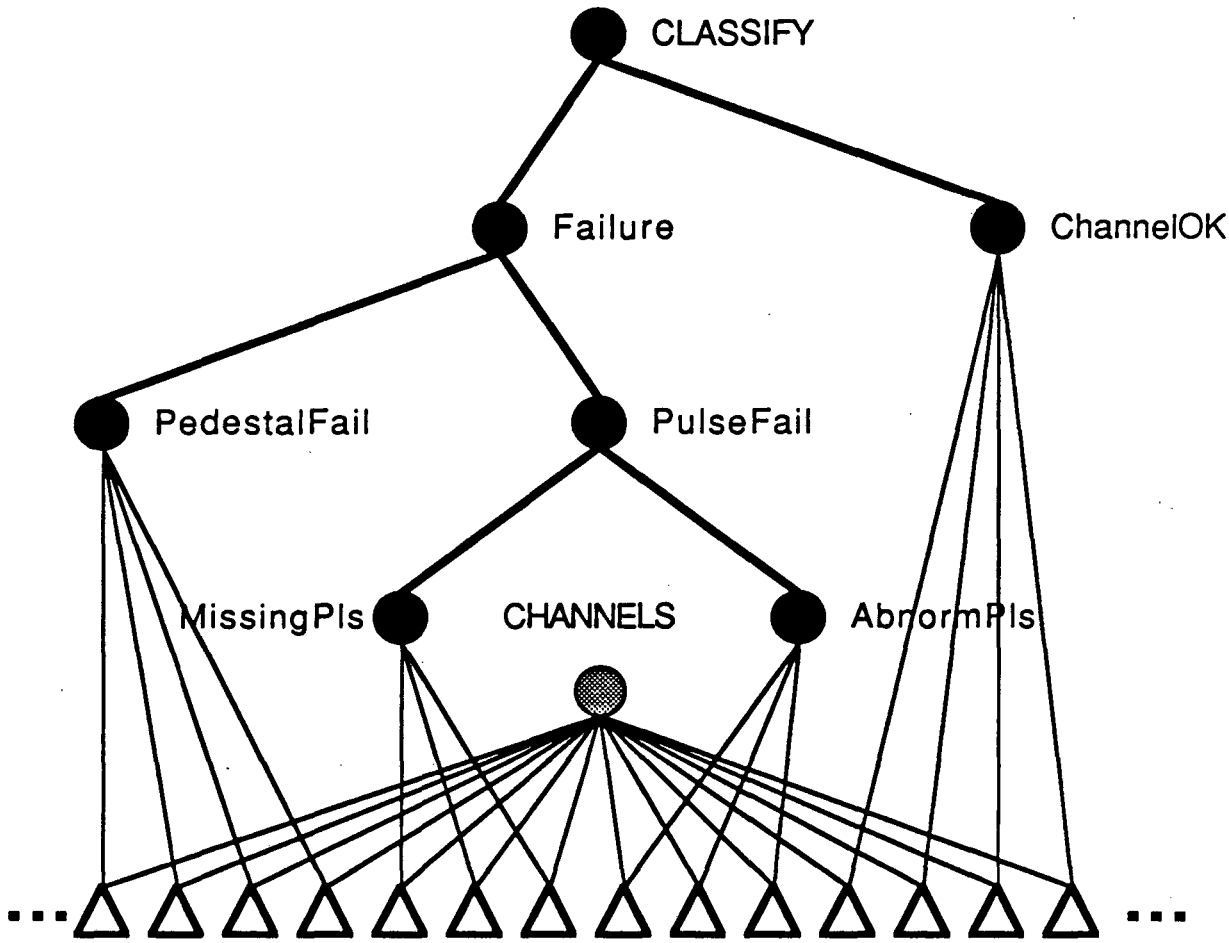


Figure 8.44: **An overview of the assignation of channels to classes.**

The final step in sorting the channels is to find which channels belonging to the Failure class or rather its sub-classes were from the same board i.e. have the same row, rack, bin and board property values. Using pattern-matching a list of channels belonging to the same group is created. Then an object is dynamically created from the class FBrds (faulty boards) and the channels contained within the list are made into its sub-objects (figure 8.45a). The channels once assigned to a faulty board are removed from the Failure class or its sub-classes and the faulty board is linked to it instead (figure 8.45b). This process is repeated until all the failed channels have been assigned to a faulty board (figure 8.45c).

The Experiential Knowledge (the Troubleshooting Rules).

Much of the information for this part of the KB has been gathered through discussions/interviews with the expert, although little has been encoded. Some of the rules are drawn up into rule trees (figure 8.46) and decision tables. There appears to be four different levels of rules from the shallow rules to deep rules.

The shallowest rules are rules that can diagnose a failure straight from the data, for example, if pulse-count is high, then change the shaper-amplifier.

The shallower mid-level rules are rules that need just a little additional knowledge to diagnose a failure, like being able to determine whether the problem is present in other channels within the same set, for example, if pedestal is missing and same problem is on all the channels in that board, then replace CCD capacitor.

The deeper mid-level rules are more general rules which need further rules to diagnose the problem, for example, if edge is missing then a board needs to be replaced; which board? To identify the problem we need another deep rule: if a problem is seen when test pulsing the shaper/amp and not when test pulsing the CCD, the failure is in the shaper/amp or the connections between. The additional information needed here is from the path of a channel KB and then additional test pulsing must be carried out.

The deepest rules are the most general rules which involve the most work to diagnose the failure after the rule has been fired. An example of this type of rule is: if the edge is high or low, then something is wrong in the hardware of the trigger system. To solve this problem it may be necessary to know about the path of the channel, the functions of the components which belong in that area of influence, about the trigger mechanism etc.

The encoding of the troubleshooting rules is the next step in the knowledge base development but was not started by the end of this year's project.

Figures 8.45: **The grouping together of channels occupying the same board.**

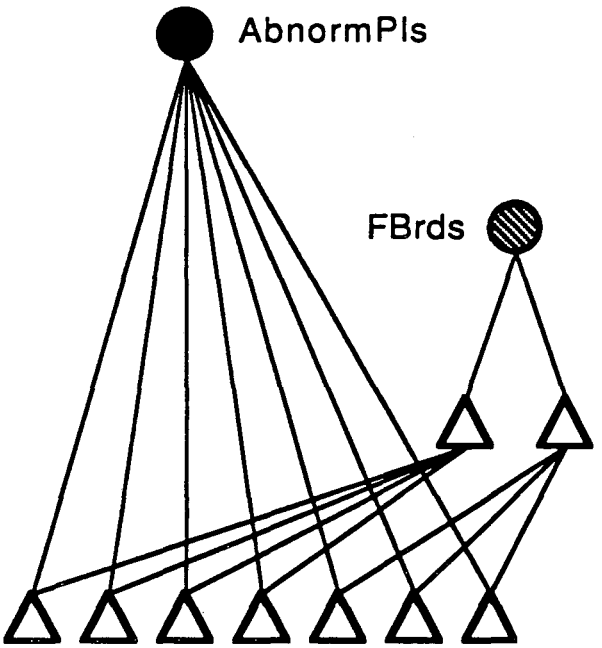


Figure 8.45a

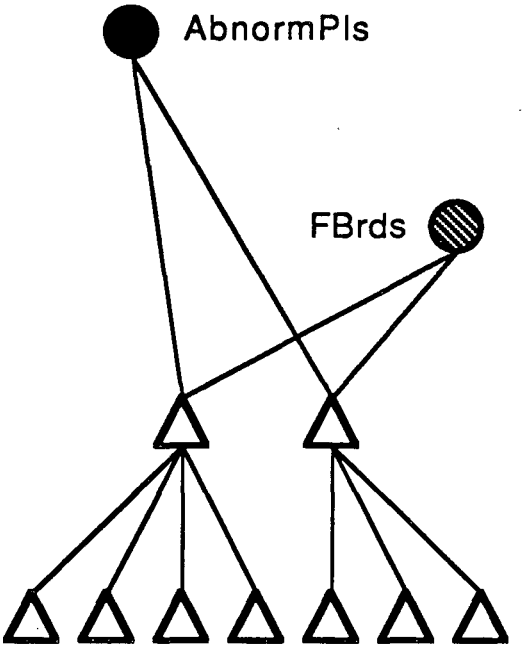


Figure 8.45b

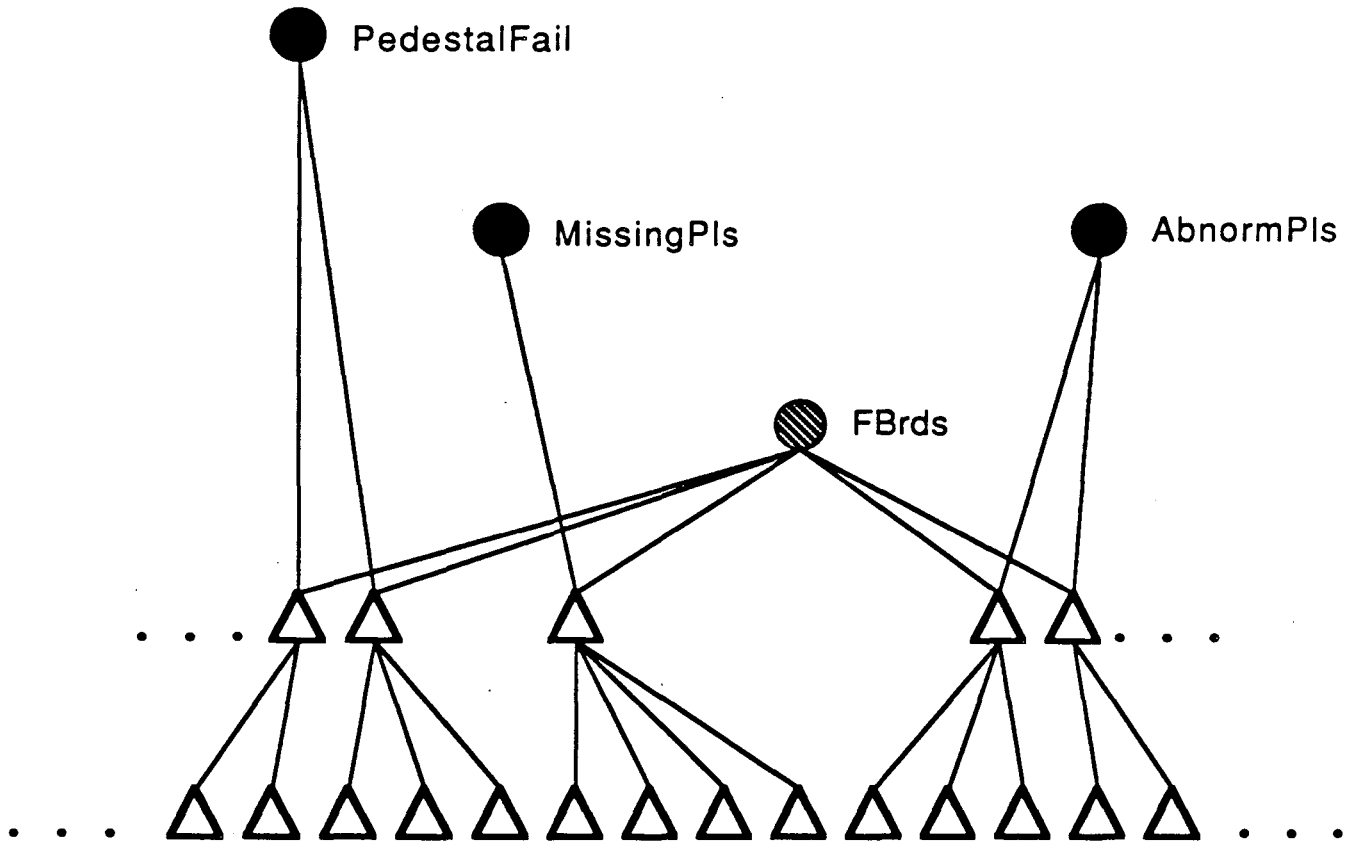


Figure 8.45c

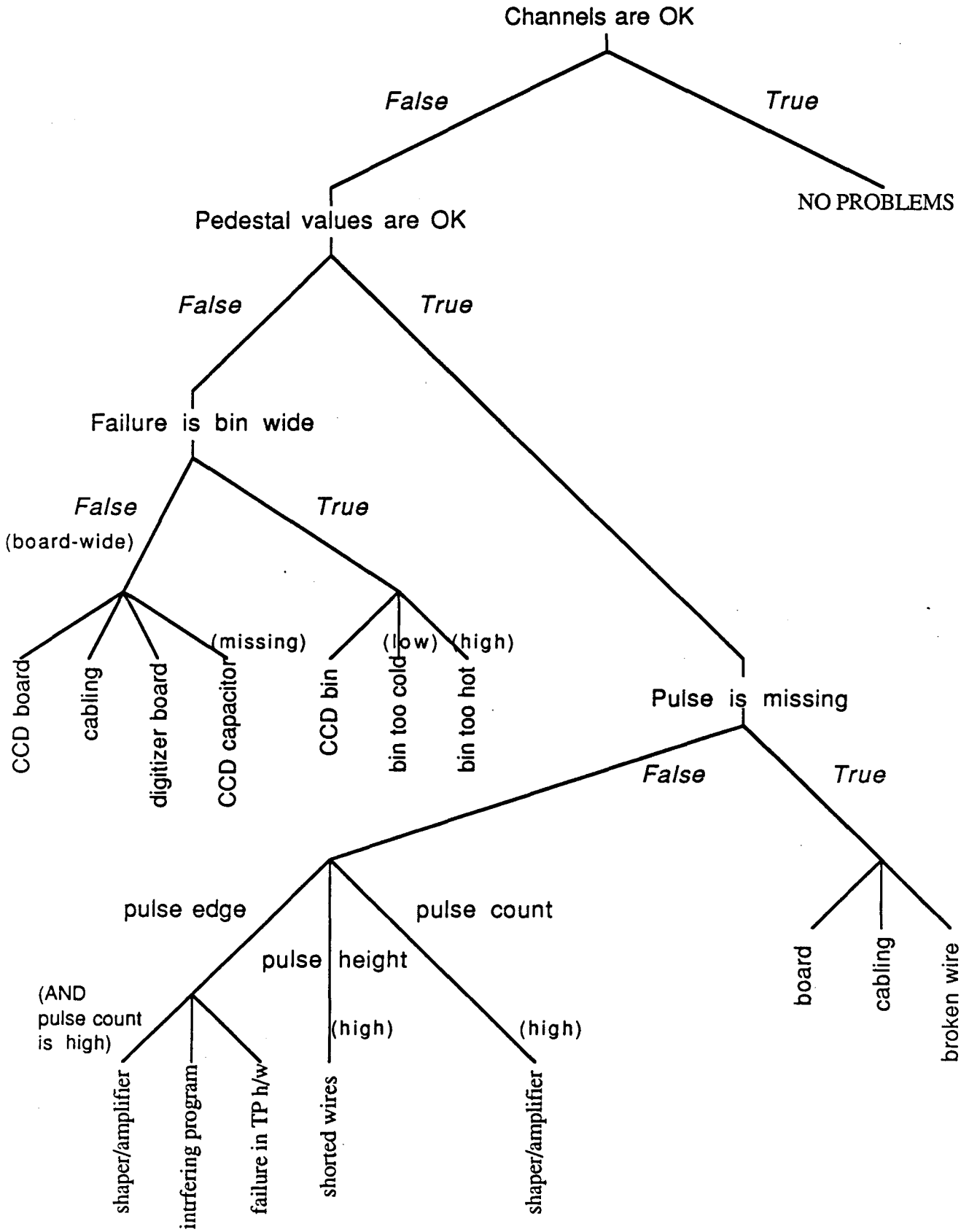


Figure 8.46: **Some Troubleshooting Rules**

9. Summaries and Conclusions.

This final chapter offers summaries of the important aspects of expert systems and of this project. It finishes with the conclusions we have drawn from this years project to demonstrate whether expert systems can provide improved solutions to a scientific research problem at LBL.

9.1 Summary of Expert System Technology.

Expert system technology is a potentially powerful technique. It provides a different approach to problems which, until now, have defied effective solution.

Expert system are computer programs that are based around heuristics, or facts and rules, rather than the algorithms of conventional programs. This enables them to deal with the imprecise and incomplete information that is part of real world problems. The inference engine and knowledge base of an expert system, are more forgiving than their conventional counterparts, the procedures and database. Expert systems also contain a certain amount of self knowledge which enables them to react to their changing environment.

The drawbacks associated with expert systems stem from their use of incomplete and imprecise data and their lack of method. These mean that expert systems are less reliable, less testable and produce only reasonable rather than optimal solutions. These disadvantages, mean that expert system technology should only be used for problems for which conventional programming cannot produce effective solutions or for those problems for which non optimal solutions are acceptable.

Expert systems are the next step in bridging the gap between what humans are good at and the things computers do well. The tasks that people are good at are: 1) converting sensory signals into cognitive ones; 2) using common sense and 3) making generalizations to enable them to draw meaningful conclusions and make useful predictions about the world around them. It is the last of these tasks that expert system technology aims to model as it attempts to work like a human expert, while overcoming some of our human limitations. People when faced with large amounts of data, will often become unsystematic, forgetful and distracted causing deterioration in the quality of their work. Furthermore, a person cannot be made to work twenty four hours a day, seven days a week, for little reward and in hazardous or unpleasant environments as a computer can. For these reasons expert systems have been gaining popularity.

Expert systems are used successfully in many fields; here we have discussed in detail only the one of troubleshooting electronic devices (in chapter 4). Expert systems are used often as intelligent assistants to remove tedious tasks from a human expert's workload and to point out inconsistencies in his reasoning when solving a problem. They are also being used to advantage for analysis and design which is particularly useful in areas where the vast number of factors involved in some of today's complex systems make it difficult for people to manage.

9.2 Summary of the Project.

In this paper we have discussed this project to investigate how expert systems can be used here at LBL. We first studied the capabilities of expert systems and then set about preparing to build a prototype expert system using a commercially available expert system shell to solve a problem of an LBL research group. After comparing a number of problems, we chose to build our system to troubleshoot the electronics of the

time projection chamber (TPC), a detector used by the high energy physicists.

We then embarked upon the knowledge acquisition phase of our prototype expert system. We studied the problem and its domain through interviews with the expert, ascertained the knowledge representation requirements, inferencing ability and interfaces with hardware, software and the user, and then selected Nexpert as the expert system shell on which to build the system. Further interviews with the expert, learning how to use Nexpert and starting the development of the data and knowledge bases took place concurrently. Questions arose from the development and more interviews with the expert were held. As we delved deeper into the development of the prototype expert system, the interviews with the expert became less frequent because finding ways to represent the knowledge already acquired was a much slower process than it's acquisition.

The stages prior to the knowledge representation went fairly much according to plan with only a few delays in ordering the Nexpert package, (due to budgetary constraints), taking the Nexpert classes (due to their scheduling) and some of the interviews with the expert who was often too busy to be able to schedule meetings at short notice.

Knowledge representation was our main stumbling block. We had difficulty representing incoming data with all it's inherent relationships. We were trying to represent, implicitly, the many channels of the TPC, only making them and their relationships explicit when they became associated with data during a consultation. Limitations in the software and our understanding of its limitations led us to modify our representation methods as we gained further experience. Our representation methods were first to represent the addresses of channels in relational dBase files (static data); then, creating a structure for the channels as they were created in the knowledge base (dynamic data - version 2); and finally, including all the necessary information in the channel objects as they were created, and only later creating a structure for the faulty channels (dynamic data - version 3). Many of the problems we encountered are discussed in chapter 8 (the building of the knowledge base for the prototype expert system) but most important were the conceptual difficulties in transforming an understanding of the TPC structure to the Nexpert software.

The exceptionally slow running of the Nexpert development package was another major problem. When asked to create more than a handful of objects, it took somewhere between five and seventy five minutes to run. This problem has been tackled in the next version of Nexpert according to Neuron Data. Because of the amount of time spent in this area of the problem, the encoding of some actual troubleshooting rules into the system never took place. We could not demonstrate our second goal of a working prototype.

9.3 Conclusions.

Our goals were two fold, first to gain experience with the currently available expert system technology and determine how it might be applied to the problems of LBL researchers, and second to build a working prototype of one such expert system.

Our first goal was certainly achieved. We gained valuable experience with expert system knowledge acquisition and representation and have a much better understanding of the types of problem that can be solved with expert systems. From our own experience and through reading about others we strongly feel that expert system technology has an important part to play in the future of problem solving, particularly those problems to do with the troubleshooting of electronic devices, at LBL. From this project we have gained a good feel for the effort and payoffs involved in the development of expert systems to troubleshoot hardware. We are now better able to judge the feasibility of future projects and set realistic goals.

There are many complex pieces of electronic equipment being used by LBL scientists for which there are no effective troubleshooting procedures. At present much of the troubleshooting capability is not computerized because there were no effective solutions using conventional computer programming methods. Many of these devices have had collections of procedural programs evolve to run non-standard tests on the electronic components. It takes an expert diagnostician for that particular device to work out which test procedures to use, when and where to use them. This can make the troubleshooting of a device unreliable and untimely because people become board and distracted while they are monitoring the device for malfunctions. Computers, using expert system technology, we believe, will be able to do the type of reasoning the human diagnostician do here at LBL at the moment, while it monitors the device consistently. Once the expert system technology has been proven to the scientific community here at LBL, it has a great potential for being used extensively, enabling us to achieve higher performance from the electronic equipment used here.

Our second goal was only partially achieved. We underestimated the difficulty that novice users of the technology would have in implementing a prototype of such a complex system as the TPC and did not complete the prototype in the allotted time. However our difficulties with the knowledge representation led us to draw three important conclusions from our attempt to build a working prototype expert system. All these conclusions, to some extent, help to explain why this second goal was not fulfilled.

The first conclusion we reached, and one that many expert system developers have commented on, is that knowledge engineering is more difficult and takes more time than we had anticipated. There is an art to representing the knowledge within a knowledge base that can be perfected only with practice. Experience in knowledge engineering, is said by Cupello and Mishelevich [2], to be crucial to the fast development of a successful prototype expert system.

Our second conclusion concerns our choice of expert system shell. Although we still believe Nexpert to have been a good choice (even more so when the promised upgrades in the next version become available), it is clear that the IBM-AT was much too slow for the system we were trying to develop. A better platform would have been the Mac II or the Sun (which was not available when we made the selection) especially as, in the advent, we were unable to call dBase programs from within Nexpert and we did not reach the stage where the Ease+ package would have been useful. A faster machine would have increased the knowledge engineer's productivity and satisfaction with the system.

Our final conclusion concerns our choice of application. We now think that the problem of troubleshooting the hardware of the TPC was too complex for our first experimentation with expert system technology. The choice of the problem rather than the choice of the sub-problem (for the prototype specification), was the mistake. If the prototype sub-problem is too limited it is not able to demonstrate the strength of the technology for that application to the users. We believe that the choice of troubleshooting hardware was a good one because it is applicable to so many situations in the laboratory, but a simpler device should have been chosen for our first attempt at expert system development.

9.4 ACKNOWLEDGEMENT.

We, the authors, would like to thank Dr. Alan Clark of LBL for finding the time to provide us with the domain expertise for this project. Without his willing help and patience much of what we have achieved would have been impossible.

APPENDICES.

Appendix A: Other Electronic Troubleshooting Expert Systems.

Here are more detailed description of the three troubleshooting expert systems that were mentioned at the end of chapter 4: FIS (Fault Isolation System) developed at the US Naval Lab., by Pipitone, in 1986, using Franz Lisp [23]; TEST (Troubleshooting Expert System Tool) developed at Carnegie Group Inc., by Kahn, Kepner and Pepper, in 1987, using Carnegie Group's Knowledge Craft (a general purpose expert system tool) [19]; and the third of these examples is AI-TEST which is being developed at Tel Aviv University in collaboration with Intelligent Electronics Inc., by Ben-Basset et al, using C (prototyping, on occasions, in Prolog) [13]. At the end of this appendix is a list of further reading, giving more examples of troubleshooting expert systems.

FIS (Fault Isolation System).

US Naval laboratory's FIS is an expert system designed to diagnose faults in analog electronic systems. It aims to isolate failures in the device to the level of larger components such as an amplifier or power supply. It is a generic electronic troubleshooting expert system which can be modified for each device by the knowledge engineer using the schematic and block diagrams and other information supplied in the documentation of the device.

FIS provides a library of common electronic components and component properties to minimize the effort of knowledge acquisition for a new device. The development, then, involves describing the connectivity and any unique components and properties. It makes use of probabilities on the fault hypotheses based on the tests used to reach them and then advises a technician on the corrective actions to take. So FIS is efficient at knowledge acquisition, it reasons quantitatively with probabilistic reasoning methods developed especially for device troubleshooting from a functional model of the device.

Knowledge Representation.

The knowledge base is made up of two main parts; the description of the device and the General electronics knowledge. The device's description is made up of connectivity information (including a graphic description of the device's block diagram); probabilities of a certain cause leading to a specified effect when a module fails; causal rules for each module clustered around the module and a set of tests which may be performed on the device along with the costs and benefits of running the them. The General electronics knowledge contains only device independent electronics knowledge in the form of a library of generic modules and module properties. The knowledge base contains no shallow model rules.

Inferencing.

An FIS session consists of creating a set of possible hypotheses and updating its beliefs with each successive test. The cost of a test, its timing and its benefits (the amount of additional, necessary information it will yield) are taken into consideration before recommending it. This system uses both forward and backward chaining in its inferencing process.

Comments.

At the time of publication of the paper (July 1986) only small devices had been tested so their were plans to scale up the applications as well as to add shallow, heuristic rules to speed up the diagnosis process.

TEST (Troubleshooting Expert System Tool).

Carnegie Group's TEST is a domain-independent diagnostic problem solver which uses an approach half way between the causal-model and the rule-based using a weak causal model to describe the causal links between failure modes and diagnostic rules to constrain and direct the reasoning. TEST includes a library of module types such as failure-modules, test-modules, etc. from which to build the diagnostic system.

Knowledge Representation.

The knowledge is represented around a frame-work of failure-modes. These failure-modes represent malfunctions in the device, they are arranged in a causal hierarchy, at the top level are the observable problems, at the bottom level are the failure-modes of the smallest replaceable components and in the middle are classes of failures and causal consequences of component failure. Failure-modes are linked to their causes (other failure-modes) by due-to links. Other modules used in TEST are questions, tests, test-procedures, rules, decision-nodes and parts (of the device). Decision-nodes are what integrate the diagnostic rules into the failure-mode oriented knowledge base. Tests are linked to other tests depending on their outcomes. Information referring to a failure-mode is clustered around it.

Inferencing.

TEST uses a depth-first, recursive strategy for inferencing, it starts with the observed or determined failures and then looks for their causes. It recommends tests in the most efficient order and allows users to volunteer unsolicited information or change previous inputs, supporting a belief maintenance system.

Comments.

TEST is now available commercially from Carnegie Group Inc. under the name TEST Bench at a cost of $42 000. Its development version only runs on the TI-Explorer but it can be delivered on an IBM-AT (the runtime version is compiled in C). They claim that it speeds up development time significantly and represents knowledge in a way that is familiar to the domain expert.

AI-TEST.

AI-TEST is also designed as a generic electronics diagnostic system building tool, its developers wished to build an expert system that would be able to troubleshoot a wide variety of devices with a minimal amount of effort to train it for each specific device. It aims to provide an intelligent service manual that guides a user through the diagnosis of a device by detecting and locating a fault down to the smallest replaceable component. The design of AI-TEST allows it to handle large scale device's and cope with both analog and digital data. It uses a functional level model to allow the expert system to reason from first and second principles.

Knowledge Representation.

The Knowledge base for AI-TEST is divided into two parts: the universal knowledge base and the device-specific knowledge base. The universal knowledge base contains all the generic electronics knowledge such as topological concepts of electronic

devices, standard modules and test measurement types arranged in a hierarchical frame-based library. This part of the knowledge base comes with AI-TEST. The device-specific knowledge base contains the information specific to that device such as its unique structure, characteristics and the tests used to troubleshoot it and is built by the user or knowledge engineer. At compilation AI-TEST checks the specific knowledge base for completeness and consistency before linking it with the universal knowledge base.

Inferencing.

AI-TEST recognizes the proper behavior of the device from the block diagram and module functions and therefore can recognize when the device is malfunctioning. It identifies possible faults and ranks them in order of likelihood and ease of testing, then recommends the most cost-effective order for the tests. The order of the tests is updated according to the results of the previous tests. Diagnosis is carried out by analyzing the test results and a priori knowledge of failure rates and characteristics of a component. To decide which modules to examine and test, AI-TEST looks at the severity of the faults, their level of uncertainty and their failure rates before choosing the next module to focus on. This is done until the smallest replaceable component is located.

Comments.

AI-TEST is seen as a software module to be embedded within a conventional test environment. It has been implemented for the following devices: military devices, test equipment, computer peripherals and communication boards (PABX); for all of these only requiring information that is available in the device documentation.

Further Reading.

Here is a list of some other diagnostic expert systems or expert system shells that may be of interest to the reader (listed in chronological order).

1. IN-ATE, developed at US Naval Research Laboratories, by Cantone et al [15].
2. NASA FIXER, developed at NASA and written about by Faught of IntelliCorp [16].
3. FBEXPERT, developed at Fermi National Accelerator Lab.s, by Booth et al [14].
4. CATS (Components, Actions, Tests and Symptoms), developed at Philips Lab.s, by Lee et al [20].
5. EMMA (Expert Missile Maintenance Aid), developed at Rockwell International Corp., by Warn [24].

Appendix B: Nexpert Questions and Answers.

Here we show some of the questions which arose while using Nexpert and the answers given by Bechtel, the suppliers, and Neuron Data, the developers of Nexpert. There were three ways to receive this information: telephone support from Bechtel (within a support contract); and two Nexpert bulletin boards, one set up by Bechtel and the later one by Neuron Data to answer users queries. Bechtel's bulletin board was reached using a remote modem, linked to the unix system at LBL. Neuron Data's bulletin board was accessed using a modem directly linked to the PC.

This appendix is appropriate for readers with some experience of Nexpert. The problems we encountered with Nexpert are categorized in three ways: problems with accessing and using the dBase files; problems with the pattern-matching construct; and other problems, such as ones with inheritance and the creation of dynamic objects.

Questions.

dBase Files.

(1) Is it possible to run dBase III+ programs from within Nexpert?

A: This can only be done if the dBase program is complied using a compiler and linker, such as Clipper (Nantucket Corp.) or Foxbase, to create a regular DOS program (changing a filename.prg to an filename.exe file) which can then be run using the execute command of Nexpert.

(2) Is it possible to search a dBase file by a field other than the one used in the one used in the object name?

A: No, not in this version of Nexpert. It is promised for the next version (version 1.1) which is now due to be released in August 1988.

(3) Can the name of the database use or include interpretations when being called in a Nexpert rule?
A: No, the database name must be explicitly called from within Nexpert.

Pattern-Matching.

(1) Can you pattern-match on the properties of sub-objects of a class of objects or the properties of the sub-classes of a class?
A: No, you can only pattern-match on the first set of OBJECTS below the class or super-object.

(2) Can you pattern-match with variable class names, or use the pattern-matching construct with interpretations?
A: No, you must put the objects of the class that you want to use into a temporary class or just refer to the class directly. This will be available in version 1.1.

(3) How do you create a class of objects, from within a larger class, with the same property values, without knowing what that value is?
A: This has to be done in two rules. So, letting the larger class be ALL_apples, the class to be created, colored_apples, with all its objects having their property, color, with the same value, that value is contained within the variable aColor. In the first rule, then, goes, on the LHS, the line: *Do <ALL_apples>.color aColor,* which puts the value of the last object in the class ALL_apples in aColor. Then in a subsequent rule, on the LHS the line: *Equal <ALL_apples>.color aColor,* and on the RHS we need the line: *CreateObject <ALL_apples> colored_apples.* Now all the apples with the color that

matches aColor are in the class colored_apples.

Other questions.

(1) Can you dynamically create an object with more than one variable in its name?
A: No, it is not possible without using external C string functions.

(2) How can you find the number of objects within a class?
A: There is no direct command for doing this. A way of finding out how many objects are within a certain class has been devised. A property, count is created within the class, CLASS. In the If Change metaslot of CLASS.count insert the line: *Do x+1 x*. Then within a Rule incorporate the line on the LHS: *Name 0 x* to initialize the variable which will be incremented. On the RHS, the line *Do 0 <CLASS>.count* does the actual incrementing. By the completion of the rule the variable x will contain the number of objects within the class CLASS.

Appendix C: Programs, Data and Knowledge Bases.

Here we describe the programs, data and knowledge bases that have been developed during this year's project and how they relate to the troubleshooting the hardware of the TPC. We will give the names of the files used and discuss them in the same order as their descriptions appear in chapter 8. We will start by naming the files used in the development of the static data, then go on to those used in the static knowledge, the three versions of the dynamic data (or input data) and finish with the file used in the pattern-matching knowledge.

i. Static Data.

The layout of the TPC is represented in dBase III+ database files and programs.

The database files (DBF) used are:

EHseLoc (Electronics House Location) - gives the location of each list of wires on endcap 0 within the electronics house.

PhysLoc (Physical Location) - gives the location of all the boards of wires in the sector given the list or visa versa.

The program files (PRG) are used with the above database files to convert one address to another. They are:

EHseLDB & LDBEHse - are used to convert between the address of a channel in the electronics house and its address in the LDB.

ECRow & RowEC - are used to find the Row a channel belongs in given the endcap is known (or visa versa).

LDBPhys & PhysLDB - convert between the LDB address and the address of the wire on the sector.

WireChan & ChanWire - convert between wire numbers and board & channel numbers.

SHOWOFF - is a program to demonstrate how one gets from any of the addressing systems used in any of the existing Fortran programs to any other.

The second part of the static data was the addressing of the test pulsing equipment. This also uses dBase III+ files. The *ITPULSER* program uses the sub-programs *TPGrid, TPPreAmp & TPShaper* and their associated databases (similarly named) to find the test pulsing equipment for any particular channel, board, bin, etc.

ii. Static Knowledge.

The file containing the static knowledge is a Nexpert knowledge base named *PATH*. It models the path that all the channels takes through the electronic components and is explained in detail in section 8.2 of chapter 8.

iii. Dynamic Data.

There are three versions of the data input discussed in section 8.3, all of them are implemented using a Nexpert knowledge base (KB).

Version1 - is the knowledge base containing the first version of the dynamic data.

Version2 - contains the second version.

TPCproto (TPC Prototype expert system) - has the final version embedded in it.

iv. Pattern-matching Knowledge.

Here we have the most complete of the Nexpert knowledge bases called *TPCproto* which is described in full in section 8.4.

v. Other Files of Interest.

There are a number of DrHalo graphical display files which are ready for use by Nexpert (compiled versions have a .np suffix otherwise they have a .cut suffix).

Begin & End - display messages for the start and finish of a consultation. There are a number of other files whose names look like this: *R0R07B2* and these files give a display of where a certain bin is located within the electronics house (the first letter stands for row followed by its number, the second stands for rack and the third for bin).

REFERENCES.

Part I.

[1] J. Aikins: A Cycle for Expert System Development...

[2] J. Cupello & D. Mishelevich: Managing Prototype Knowledge/Expert System Projects. May 1988, vol 31, No 5, Communications of the ACM.

[3] P. Denning: Towards a Science of Expert Systems. Summer 1986, IEEE Expert.

[4] R. Forsyth: Expert Systems - Principles and Case Studies. 1984, Chapman and Hall Computing, UK.

[5] B. Hancock: Expert Systems. May 1987, DEC Professional.

[6] P. Harmon & D. King: Expert Systems. 1985, Wiley Press NY.

[7] F. Hayes-Roth, D. Waterman & D. Lenat: Building Expert Systems. 1983, Addison Wesley Publishing Company, Reading MA.

[8] F. Hayes-Roth: Rule-based Systems. September 1985, vol 28, no 9, Communications of the ACM.

[9] W. Rolandi: A Practical Approach to Knowledge Engineering. April 1986, AI Expert.

[10] D. Waterman: A Guide to Expert Systems. 1986, Addison Wesley, Reading MA.

[11] C. Williams: Expert Systems, Knowledge Engineering, and AI Tools - An Overview. Winter 1986, IEEE Expert.

[12] I. Zualkernan, W. Tsai & D. Volovik: Expert Systems and Software Engineering: Ready for Marriage? Winter 1986, IEEE Expert.

Part II.

[13] M. Ben-Basset, D. Ben-Arie, I. Beniaminy, J. Cheifetz & M. Klinger: AI-TEST A Real Life Expert System for Electronic Troubleshooting (A Description and a Case Study). 1988, IEEE 4th Conference Proceedings for the C.A.I.A.

[14] A. Booth, J. Carroll & G. Goerannson: FBEXPERT - An Intelligent Assistant to Diagnose FASTBUS. August 1984, vol ns-34, no 4, IEEE Transactions on Nuclear Science.

[15] R. Cantone, F. Pipitone, W. Lander & M. Marrone: Model-Based Probabilistic Reasoning for Electronics Troubleshooting. 1983, Joint Conference on AI.

[16] W. Faught: Applications of AI in Engineering. July 1986, Computer.

[17] P. Fink, J. Lusth & J. Duran: Expert Systems Diagnostic Expertise in the Mechanical and Electrical Domains. May/June 1987, vol smc-17, no 5, IEEE Transactions on Systems, Man and Cybernetics.

[18] P. Fink, J. Lusth & J. Duran: A General Expert System Design for Diagnostic Problem Solving. September 1985, vol pami-7, no 5, IEEE Transactions on Pattern Analysis and Machine Intelligence.

[19] G. Kahn, A. Kepner & J. Pepper: TEST: A Model-Driven Application Shell. July 1987, 6th National Conference on AI, Seattle WA.

[20] K. Lee, J. Martin, P.Rutter & R. Wexelblat: A Development Environment for Field Diagnosis Tools. 1988, 4th Conference Proceedings of the CAIA.

[21] B. Merritt: Anatomy of a Diagnostic System. September 1987, AI Expert.

[22] R. Milne: Strategies for Diagnosis. May/June 1987, IEEE Transactions on Systems, Man and Cybernetics.

[23] F. Pipitone: The FIS Electronics Troubleshooting System. July 1986, Computer.

[24] K. Warn: EMMA Expert System Diagnostic System...

Part III.

[25] R. Citerenbaum, J. Geissman & R. Schultz: Selecting a Shell. September 1987, AI Expert.

[26] W. Gevarter: The Nature and Evaluation of Commercial Expert System Building Tools. May 1987, Computer.

[27] D. Gross: PC-Based Shells Let Users Design Expert Systems. March 16, 1987, InformationWEEK.

[28] B. Olsen, B. Pumplin & M. Williamson: The getting of wisdom: PC expert system shells. March 1987, Computer Language.

[29] S. Shepard: Simplifying the shell game. January 1987, Computer Language.

[30] J. Somsel: NEXPERT Object and HUMBLE: Object-Based Shells. November 1987, AI Expert.

[31] L. Wilson: Expert Systems and C: CxPERT. May 1987, AI Expert.

LAWRENCE BERKELEY LABORATORY
TECHNICAL INFORMATION DEPARTMENT
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 94720