A Prototype Expert System Based Diagnostic Tool for Cable Trunk Amplifier Networks

by

Stephen William Neville
B.Eng., University of Victoria, 1990.

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF APPLIED SCIENCE

in the Department of Electrical and Computer Engineering

We accept this thesis as conforming
to the required standard

Dr. N.J. Dimopoulos, Supervisor (Department of Electrical and Computer Engineering)

Dr. K.F. Li, Departmental Member, (Department of Electrical and Computer Engineering)

Dr. Z. Dong, Outside Member, (Department of Mechanical Engineering)

Dr. I. Morrison, External Examiner, (Acquired Intelligence)

University of Victoria

Supervisor: Dr. N.J. Dimopoulos

Abstract

As the complexity of communication networks increases, it becomes increasingly difficult to isolate and diagnose network faults. This maintenance problem is due to the large number of functional elements within the networks, the elements' technical complexity, and their inherent interaction. One technique to solve this problem is to use expert system technology to manage the vast amount of status information produced by these networks and to provide an automated diagnostic reasoning about the location and cause of observed faults. In this thesis, a prototype expert system based diagnostic tool for cable trunk amplifier networks is presented. This tool is capable of isolating and diagnosing the most common network level faults which occur in these type of networks. This tool was created as a proof-of-principle of the use of expert system based diagnostic tools on this type of communication network. As such, the tool was tested on a series of network faults occurring within the Rogers Cablesystems Ltd. Victoria cable trunk amplifier network. The results of these tests are presented.

Examiners:

Dr. N.J. Dimopoulos, Supervisor (Department of Electrical and Computer Engineering)

Dr. K.F. Li, Departmental Member, (Department of Electrical and Computer Engineering)

Dr. Z. Dong, Outside Member, (Department of Mechanical Engineering)

Dr. I. Morrison, External Examiner, (Acquired Intelligence)

Table of Contents

List of Tables

Table of Figures

Acknowledgments

I would like to acknowledge the contributions, academic or otherwise, of the following people:

Dr. N.J. Dimpoulos, for guiding the research and seeing it through to completion,

Mr. John Foss and Mr. Bill Massey of Rogers Cable Television Ltd., Victoria for providing resources and information needed for this research.,

Special thanks to Mr. John Anderson, Research Engineering, Rogers Cable Television Ltd., Toronto, for his assistance in explaining the operation of the cable trunk amplifier networks,

Canadian Cable Labs Fund, and its director Dr. John Madden for making this research possible,

and, finally, my wife, Amanda.

Chapter 1:

Network Diagnostic Tools

1.1 Introduction

Over the past several decades the complexity of communication networks has grown considerably. This is the result of many factors including the creation of physically larger networks, the increased functionality of network components, the error correction ability of some components, and the creation of heterogeneous networks to name a few. The error correction ability common in many conventional networks creates particular problems in fault diagnosis since the network may be able to function marginally even though a network fault exists. In effect, this error correction ability masks network problems in some cases. This increased complexity has made the process of diagnosing network faults a much more difficult and much less efficient task [1][2][3].

To be able to efficiently isolate and diagnose a fault, the operator must have detailed understanding of each type of equipment used in the network and how they interact, as well as an intuitive "feel" for which type of information will be important for diagnosing the current fault. This type of expertise is usually gained only after the operator has had several years of experience in maintaining the specific network. Typically, operators without this expertise will be overwhelmed by the amount of information that must be referenced in order to be able to determine the state of the network and, hence, to be able to diagnose the fault. Inexperienced operators will also perform much less efficiently, particularly for novel faults [4], than the experienced network operators. As network complexity continues to increase, even the experienced operators will start becoming overwhelmed.

One approach to solve this problem is to use expert systems technology to manage and perform diagnostic reasoning on this vast amount of network information. The resulting diagnostic tool's performance can then be evaluated by comparing it to the performance achieved by the domain experts. This technology utilizes techniques from the field of artificial intelligence to allow computer programs to operate in manner similar to human experts within a given domain. Over the past several years, a fair amount of research has been done on the theory behind how these network diagnostic expert systems should be created. The following section of this chapter will present an overview of four particular examples of network diagnostic expert systems. Each of these expert systems operates on a different type of network and, hence, are structured slightly differently to provide solutions to the issues faced in diagnosing that particular network. These examples will be presented both to introduce the theory behind network diagnostic expert systems and to present concrete examples of the network diagnosis problem. This overview will then be followed by a summary of the salient features of these diagnostic tools. A brief introduction to the particular problem addressed by this thesis will then be presented along with an outline of the remaining chapters.

1.2 Example Network Diagnostic Expert Systems

The following four network diagnostic expert systems each operate on a different type of communication network, from a digital radio network, to a telephony switching network, to heterogeneous computer LAN, and, finally, to an analog telephony special services network. Each of these networks present unique types of faults that must be diagnosable by the associated expert system. Even so, these diagnostic tools all contain many of the same general features that are needed to have a robust expert system within the domain of network fault diagnosis, although their implementations vary across the four tools. These discussions do not included detailed descriptions of how the knowledge possessed by the domain experts was captured and incorporated into the various diagnostic tools since this information was not available from the research groups' publications.

1.2.1 Digital Radio Telecommunications Diagnostic Expert System [5]

Commercial digital radio networks consist of a large number of remote transmission sites, each containing multiple independent transmitters and associated equipment, which are separated over a large geographical area. The network is monitored by a central office which polls each transmitter at each transmission site about its current status. If an "unhealthy" state is returned by a particular transmitter then the central office requests the faulty transmitter to provide a detailed status report. Each radio has approximately 30 possible reportable faults, so over a typical network consisting of several hundred receiver and transmitters there are over ten thousand possible reportable malfunctions. Complicating the diagnostic process is the fact that a given radio's function within the network may be changed by commands generated at the central office either automatically or manually. The main goal of this diagnostic expert system was to take this vast amount of network information and use it effectively and efficiently to produce real-time fault diagnoses equivalent to those produced by an experienced engineer.

To reach this goal, the expert system performed diagnosis through three progressive layers emulating the techniques used by experienced field engineers. The first two layers isolated the fault to a particular transmitter or transmission site within the network, while the final layer performed a detailed diagnosis of the faulty network element. When faults occurred within adjacent transmission sites, the diagnosis layer employed was termed *adjacency detection* and its goal was to flag the component at the head of the signal path as the faulty element. When an error effected only one transmission site, *co-location detection* was used as the diagnosis layer and its goal was to flag all the same site components that were in an alarm state as faulty. The final layer, if time permitted, would try to determine a detailed cause of the alarms attributed to flagged components.

The expert system performed its diagnoses through modeling the structure and behaviour of the physical network. The network's structure was represented within the traditional objects, attributes, and value system common in expert systems. Classes were created to describe the types of components that existed within the network. Particular components were then instantiated as objects within the given class that described the particular type of component. The object's attributes were then set to reflect the physical properties of the given component. The behaviour of the network was modeled through the alarm state of the network components. This alarm state was represented as a class within the knowledge base instead of being an attribute of the component objects. This novel feature allowed the representation of multiple alarms occurring on a single device and it also allowed recommendations to be made for all the alarms even if they occurred on different components.

The final diagnostic expert system was able to efficiently and effectively diagnose the digital radio network by applying the same disconsolate rules as the experienced field engineers. In addition to the above, some of the other important features of this diagnostic system were:

- Knowledge of multiple experts was used in the development of the rule base.
- Data from the actual real time environment was used in the reasoning process and the depth of reasoning was determined by available processing time.
- More weight was given to certain types of data when performing the fault isolation.

1.2.2 Network Operations and Maintenance Centre (NOMC) [6]

This expert system operated in the domain of telephony switching networks, specifically networks developed around Fujitsu's FETEX-150 digital switching system. Unlike the previous system this tool is concerned with both the operation and maintenance of the network. In terms of operation, the tool needed to be able to perform flow control tasks, such as re-routing to avoid congested areas, automatically. The maintenance section was quite similar to the previous example in than it dealt with locating and diagnosing faults, although the methods it used to perform this task are considerably different. Since this thesis is focused on network fault diagnosis, the operations section of this system will not be discussed in detail.

The main focus of this system is to help operation and maintenance (O&M) technicians make optimum decisions. This expert system employs a two step diagnostic hybrid reasoning mechanism to isolate the cause of a particular fault. The first step involves using deductive reasoning to attempt to isolate the given fault[7]. If this process cannot locate the faulty part then hypothetical reasoning is employed[6]. Figure 1.1 illustrates the steps that were used to accomplish this type of reasoning.

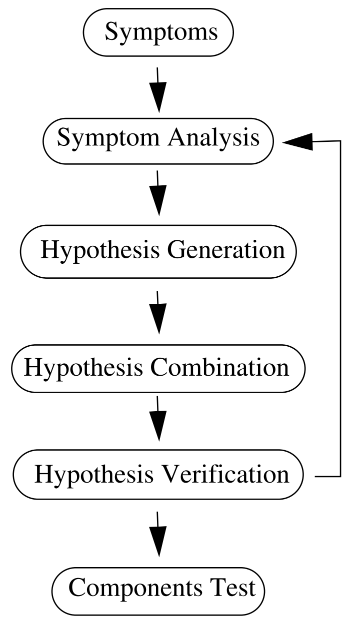


Figure 1.1: Steps in the Hypothetical Reasoning Process[6]

The first step in this process was to collect the symptoms of the fault from the switching network in order to determine the function that the network was executing when the fault occurred. Hypotheses were then generated about the cause of the fault by tracing the signal path that was in effect when the given function was being executed. For example, if the network was attempting to connect two distant nodes when the fault occurred, then a hypothesis might be generated suggesting that the cause of the failure was a given intermediate node. If this hypothesis was selected for testing, then the system would actively query the suspect node in order to determine its operational status. The system was capable of generating multiple selective hypotheses from only singular symptoms. One hypothesis was then selected for each symptom and these hypotheses were combined to obtain a group of the possibly faulty components that were capable of producing all of the observed symptoms. This group of components was then presented to the technician, who then determined if the components had actually failed. If the components were not actually in failure then another hypothesis was selected and the process was repeated until the faulty components were isolated.

One interesting feature of this process was that the technician was actively involved in selecting and verifying hypotheses. This feature allowed this system to exploit human diagnostic capabilities while still using the automated system to narrow the field of interest, thereby, not overwhelming the O&M technician. Studies have shown that as problem size increases humans increasingly deviate from the optimum solution path [3]. This system attempted to stay on this optimum path by allowing the expert system to focus the search for the solution, which in this case was the isolation of the faulty network component.

The knowledge base was structured using the object oriented paradigm. In this hierarchical structure, the switching system was defined as the top layer with the subsystems, such as speech and control paths, located underneath it and the bottom level representing actual hardware components, such as packages and cabling. The interconnection of components within the network was represented as links between these various objects. Since this expert system was designed to diagnose different versions and physical installations of this particular type of network, the knowledge base was divided into switching configuration knowledge, version specific knowledge, and common knowledge. To be able to diagnose a particular network a tailored knowledge base was first constructed by combining the common knowledge, the specific knowledge for that version of the switch, and the specific knowledge about how that switch had been configured when it was installed. The expert system also had the ability, through a knowledge base management subsection, to add new diagnostic information obtained by correcting novel faults to the knowledge base.

The knowledge base was also designed to be maintained by the O&M technicians themselves, and not by experts in the diagnostic tool. This entailed building an interface to the knowledge base through which the technicians could operate using terms understandable to them, as opposed to the cryptic languages commonly used in most expert systems. For example, the structural knowledge was represented to the technicians in a graphical form which they could easily modify or verify. For processing efficiency an internal knowledge representation also needed to be maintained, so this interface acted as a translator from internal to external representations and vice versa.

This system was able to correctly diagnose faults within the switching network with an 80% accuracy. The researchers expect that the final version of the systems, when fully implemented, will have a diagnostic capability in excess of 90%.

1.2.3 LAN Diagnostic Expert System (LODES) [8]

This expert system operated in the domain of complex, heterogeneous computer networks employing the Transport Control Protocol /Internet Protocol (TCP/IP) family of communication protocols. The main goal of this network was to isolate and diagnose faults caused by communication protocol violations or physical network failures in a timely manner. The system was not designed as an all-round troubleshooter, but instead it was designed to diagnose common network faults quicker than human technicians. This freed the technicians to work on the diagnosis of the more complex and novel network faults, something that humans are better suited to than general expert systems [9][10]. The heterogeneous nature of these networks makes the task of fault isolation particularly difficult. The diagnostic system must be able to perform fault processing based on knowledge of all the types of computers on the network and their interconnections.

This processing can be done either by having one central knowledge base containing all of the relevant information about all the computers, or by having a distributed system in which information about local computers is stored on the local computers themselves and diagnoses are performed cooperatively over the network. The former method tends to result in an inefficient and inaccurate diagnostic system since all the information must be collected in a central location, necessitating maintaining copies of some of the information located locally on the computers, and the processing must occur at this central location. There is a high overhead needed to maintain the consistency of this copied information and centralizing the processing can result in a high network load at the node associated with the diagnostic system. This high loading could cause some network faults to occur in their own right. The latter method has the advantages that the processing is distributed so no one node is the focus of the communication required for the diagnostic processing, and that there is no need for consistency checking since copied information no longer needs to be maintained. The main disadvantage of the distributed approach is that the control structure of the diagnostic tool becomes considerably more complex.

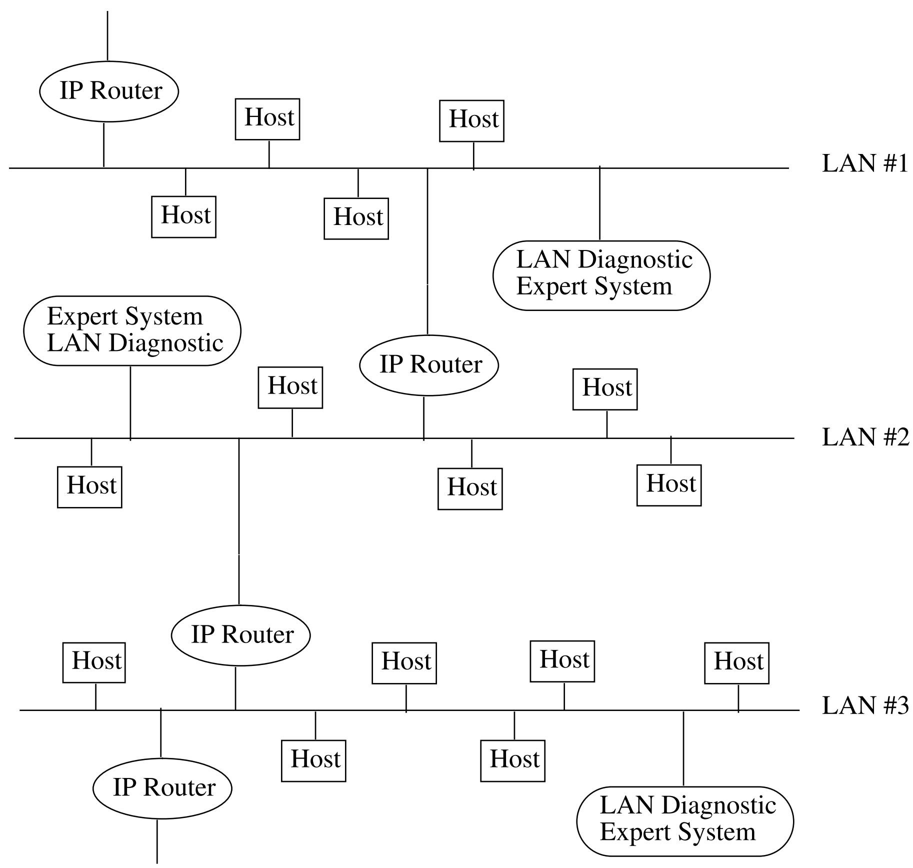


Figure 1.2: Heterogeneous computer communications network with distributed diagnostic expert system.

For this particular system distributed diagnostic processing was implemented by having an expert system located within each homogenous subsection of the heterogeneous network (Figure 1.2). These local LAN expert systems were responsible mainly for diagnosing faults that occurred within the local network. If a fault was determined not to belong to the local network, either because of the type of symptoms exhibited or because no local cause could be found, then the local expert system would request diagnostic information from other LAN expert systems in the heterogeneous network. For example, if a communication problem was reported between a host on LAN #1 and a host on LAN #2 of Figure 1.2, then the expert system on LAN #1 may have needed to acquire diagnostic information from the expert system on LAN #2 to isolate the fault. The two expert systems would then work cooperatively to arrive at a diagnosis of the network fault.

The majority of the knowledge base of each of the LAN diagnostic expert systems was concerned with information about fault diagnosis within the local network itself. This diagnostic knowledge was divided into two parts termed *control knowledge* and *investigation knowledge*. The control knowledge was used to select one item as the possible fault location and pass this item on to the investigation knowledge which was used to check the validity of this fault hypothesis. If the hypothesis was proven invalid by the investigation knowledge then a new item was selected from a blackboard by the control knowledge and the process was repeated. If the hypothesis was proven correct then the faulty element had been located. In addition to this local diagnostic knowledge, the knowledge base also contained information about IP addresses and routing tables for all the hosts attached to the particular local network, information about TCP/IP and upper level protocols, and information about normal network status, which was used in the detection of abnormal network operations. Because of the relative complexity of cooperative diagnostic processing, knowledge was also stored about how to request diagnostic information from the other expert systems and how to service such requests. Finally, the knowledge base also contained information about how to collect and analyze transmission packets from the network. This specialized knowledge was needed in order to efficiently select packets for analysis from the very high traffic levels common over the network.

The final system was able to diagnose common network faults typically within under one minute. This is far faster than human experts were able to diagnose the same type of faults. As such, this system achieved its goal of freeing technicians to work on solving complex and/or novel faults for which the system could not provide a diagnosis. This system also mimicked the cooperation employed by technicians to isolate inter-network faults by utilizing a cooperative diagnostic procedure between the two affected LAN diagnostic expert systems.

1.2.4 Expert Tester (ExT) [11]

This diagnostic expert system was developed to operate in the domain of special service analog telephony circuits. These circuits typically have a wide variety of configurations and diverse components and, therefore, they represent a very difficult class of networks on which to perform fault diagnosis. This domain was chosen since it addresses the most complex circuits used by Bell Telephone companies. Therefore, a system which provides accurate diagnoses on these circuits should function equally well, if not better, on less complex circuits, provided the appropriate knowledge base is employed. Like the previous systems, this system was designed to perform diagnoses in a similar fashion to expert technicians by employing a layered knowledge base.

This layering was accomplished by using behavioural, structural, and functional modeling of the network elements. The structural modeling described the interconnection of the various network elements and the relationship between components within each element. The behavioural modeling specified the mapping between the input and output of each of the network's elements and the elements' components. The functional modeling described the intended purpose of each of the elements and components. These models were implemented using the object oriented paradigm as slots within the particular element or component's frame. This is basically identical to the use of objects and associated attributes described in the first example if the frame is considered as the object and the slots are considered to be the object's attributes. For each element and component within the network there exists an associated frame within the knowledge base. In addition to this modeling information, each frame also contains slots for information regarding allowed operations, failure rates, and a knowledge base instantiation. The allowed operations slot details the types of tests that are valid for the given piece of equipment and the corrective actions to be taken when this equipment is found to be faulty. The failure rate slot is used to describe the a priori probability, derived from previous fault diagnoses, that the given equipment has failed. The instantiation slot contains information regarding how the particular type of equipment is to be instantiated within the knowledge base.

ExT requires that all the knowledge to be processed by the system exists as instantiated frames. For this reason, the various tests that may be performed by ExT are also represented as frames within the knowledge base. These test frames contain information about the preconditions needed to perform the test, the post conditions resulting from the test, the test results, and the command sequence required to perform the test. Once a test is performed, conditional probabilities associated with the test are used to modify the failure probabilities associated with each of the network elements. The use of conditional probabilities allows the system to function adequately without the need for an exhaustive set of rules governing how the belief in the correct operation of any network element is effected by the results of a particular test.

The first step in ExT's diagnostic process is to build the network model within the knowledge base from a circuit description passed to the system by an independent circuit administration system. This building process is aided by the instantiation information contained within the prototypical frames for each type of network equipment. This process creates a hierarchical abstract description of the network with the major subsystems represented at the top level and the discrete circuit components represented at the lowest level. This hierarchical structure is used within the fault diagnosis process to successively relax underlying assumptions by progressively moving from hypotheses concerning top level elements to hypotheses concerning the low level elements.

When a misbehaviour is detected in the network state, ExT can proceed along one of two paths. If there is enough initial information, ExT can begin generating hypotheses about the possible cause of the fault. If the initial information does not contain enough detail, then ExT can perform “monitor” testing in order to collect additional information. The tests that are employed in this monitoring phase come from the general set of available tests. The system selects particular tests form this set by looking at the failure rates for the various network components and previous test results. For example, if a particular component has a high failure rate then its associated test would have a higher chance of being used within the monitoring phase. The results of these monitoring tests can then be used to generate the initial set of plausible hypotheses. If initial symptoms are present then a special routine can generate certainty factors for each of the network elements.

Once this set of initial hypotheses is generated, ExT then selects a small subset of the more plausible hypotheses from this set. A decision about which of the allowed tests will provide maximal discrimination between these hypotheses is then taken and the resulting tests are applied. This information concerning the discriminatory power of the various tests may already be known by the system or, alternatively, it can be generated from the structural model of the network. Once this set of tests has been performed, the resulting information is used to further reduce the set of plausible hypotheses. ExT now uses the behaviour model to further reduce this set. By determining the expected outputs of the elements within the network for the given network state, ExT can eliminate hypotheses concerning elements which would have been incapable of generating the fault. The final hypothesis reduction step is performed using the functional model of the network. Any hypotheses that are left in the set after this reduction are considered to be the cause of the network misbehaviour. As the set of plausible hypotheses is modified, the certainty factors associated with each network element are also modified accordingly. These factors are also used to direct the fault diagnosis.

The current version of ExT has been deployed within the Special Services centers of the Regional Bell Operating Telephone Companies and has been successful in diagnosing faults within a number of special service circuits.

1.3 General Principles for Network Diagnostic Tools

From the above examples, it can be seen that there are some key features required to produce expert systems that can operate in the domain of network fault diagnostics. Some of these features were common to all of the example systems while others were only implemented within a particular system. This difference is due to the properties of the particular networks that the systems were developed to diagnose. These different network properties lead the system designers to make different choices about which of the diagnostic tool features to implement within their system. Therefore, a particular feature may be of general interest to the field of network diagnostic tool development without necessarily being present in all such systems. The following is a list of the desirable network diagnostic tool features obtained by analyzing the previously presented examples.

- The system should utilize structural, behavioural, and functional modeling to provide deep reasoning capabilities. This creates a more powerful tool which is capable of dealing with a wider variety of network faults than would be possible with traditional expert system utilizing only shallow reasoning.

- An object-oriented paradigm should be used to represent the various models of the network since this paradigm is well suited to representing interconnected abstract objects.
- The system should be able to collect information about the state of the network and diagnose the fault in real-time. This is particularly important in the use of the system to diagnose common faults and, thereby, allow the technicians to concentrate on complex, novel faults.
- The system should perform its reasoning in layers. As the search for the fault narrows, progressively lower level knowledge bases should be employed. This allows the system to work in a progressive manner similar to that used by human network experts when they perform top-down analysis. In a real-time environment, this also allows the curtailing of the depth of a given diagnosis due to time constraints.
- If possible, the system should work cooperatively with the human experts in order to achieve high diagnostic accuracy and efficiency for both routine and novel faults.
- Where possible the system should also incorporate the diagnostic knowledge and processes used by experienced technicians since this will help to improve the system's overall diagnostic accuracy. This allows for the expert system to combine both the deep reasoning provided by the theoretical modeling and the shallow reasoning provided by the domain experts' experiential knowledge.
- Finally, in heterogeneous networks the use of a distributed diagnostic tool may be applicable if the overhead associated with the timing of the distributed components is not excessive for the particular network.

1.4 Thesis Description and Outline

The work involved in this thesis arose out of a desire of Rogers Cablesystems Ltd., through the Canadian Cable Labs Fund, to improve the maintenance of their cable television trunk amplifier networks. These networks consist of between several hundred to a couple thousand bi-directional, cable amplifiers interconnected in a tree structure. These networks are used to transmit the cable television signals from a central source to distribution amplifiers and from there into the subscribers' homes. Specifically, this thesis details the development of a prototype expert system based network diagnostic tool that is capable of isolating and diagnosing a number of different types of network faults that occur in Rogers’ Victoria cable trunk amplifier network. The thesis is divided into three major parts. The first part presented an introduction to the theory behind network diagnostic expert system tools. The second part describes the current structure of Rogers’ Victoria cable amplifier plant, some of the fault modalities of this network, and the current diagnosis procedures employed by Rogers’ personnel. The final part details how the network analyzer was constructed both in terms of its knowledge base and knowledge processing.

This thesis is divided into the following chapters:

- Chapter 1 (this chapter) provided background theory relating to the development of network diagnostic expert systems.
- Chapter 2 details the structure, some of the fault modalities, and current fault diagnosis procedures employed by Rogers on their Victoria cable trunk amplifier network.
- Chapter 3 details how the network topology was represented on-line and the additional software tools that were needed to provide the network diagnostic tool with access to the various information sources.
- Chapter 4 details the knowledge processing performed by the tool by describing how faults were isolated, and how the expert system’s knowledge and rule base were structured to diagnose the fault once the fault isolation was performed.
- Chapter 5 highlights some of the fault diagnosis results that were achieved using the network analyzer on data collected for the Victoria cable network.
- Chapter 6 presents a summary of the conclusions of this thesis as well as recommendations for future work.

A number of appendices are also included, covering a variety of topics.

1.5 Trademarks

There are far too many references to specific software products and network equipment within this work to note the associated trademark as each is used. Therefore, the following is a list of the trademarks used within this work and their associated owners.

X and **X Window System** are trademarks of M.I.T.

OpenWindows is a trademark of Sun Microsystems Inc.

UNIX is a trademark of American Telephone and Telegraph, Inc.

Sun, **Sparc** and **SparcStation** are trademarks of Sun Microsystems, Inc.

NEXPERT OBJECT is a trademark of Neuron Data, Inc.

C-COR and **C-COR Electronics** are trademarks of C-COR Electronics, Inc.

FLOWTOOL is a trademark of the University of Victoria.

Chapter 2:

Rogers' Victoria Cable Amplifier Network

2.1 Introduction

Cable television distribution networks, commonly termed cable plants, are used to distribute the cable signals from a centrally located site or sites within a given city to the individual subscribers' homes. It is the responsibility of these networks to provide consistent and high quality signals to all of the cable subscribers. The basic structure of the networks consists of several hundred to a few thousand cable trunk amplifiers interconnected by coax cable spans in a tree like organization. The actual number of amplifiers depends on the number of subscribers to be serviced and signal propagation concerns. At the leaf nodes of this tree, there are smaller spans of distribution amplifiers which propagate the cable signals from the main cable trunk network down into local neighborhoods. Taps from these distribution amplifiers then feed signals to the cable subscribers' homes. A number of taps may be used at each distribution amplifier, allowing the amplifiers to, effectively, provide cable service to two or three hundred subscriber's homes a piece. There may be ten to twelve such distribution amplifiers supplied by a single trunk amplifier, so a failure in one trunk amplifier may affect many hundreds of cable subscribers. Typically, though, a fault in one trunk amplifier will be propagated down the network causing many more subscribers to be affected. The detailed physical properties of a particular cable plant is determined by the actual type of amplifiers that are utilized.

This chapter describes the structure of Rogers Cablesystems Ltd.'s Victoria cable amplifier plant both in terms of the physical properties of the network and in terms of the current procedures that are employed to perform network maintenance. Since Rogers utilizes amplifiers produced by C-COR Electronics Inc., this description will be applicable only to cable plants using this specific amplifier technology. In practice, though, this technology is used throughout Canada by Rogers Cablesystems Ltd. and by many U.S. cable companies. The description of this cable plant will proceed by first describing C-COR's trunk amplifiers. The actual layout of the Victoria cable plant will then be discussed. The monitoring and maintenance procedures for this cable plant will then be presented. Finally, the failure modalities typical of this type of cable plant will be presented. No attempt will be made to present a description of the distribution amplifiers directly since they are not currently included within the fault model of the diagnostic tool.

2.2 C-COR Cable Trunk Amplifier

The amplifiers used in the Victoria cable plant are produced by C-COR Electronics Ltd. and are specifically designed for the cable television industry. These amplifiers are capable of amplifying signals over a range of 50 to 550 MHz, providing a full two-way communications capability, and have a completely modular design to improve maintenance. This two-way communication is divided into a forward path and a reverse path. The former is used to distribute the cable T.V. signal from the head end of the network to the subscribers, while the latter is used mainly in the collection of network status information.

The remainder of this section will describe the functionality of these various modules by presenting the amplifier's theory of operation. This discussion is not meant to fully present the features and operational characteristics of this particular family of amplifiers, but instead to provide the functional details necessary to understand the various failure modalities suffered by networks employing this specific amplifier technology. Since the current diagnostic tool is only concerned with network level faults, no attempt will be made to discuss modules or module information which does not contribute to an understanding of the amplifier's operation at this level. Specific values for various electrical signals that are mentioned within this section refer directly to the Victoria cable plant and may not be indicative of the general signal levels employed in cable plants utilizing C-COR amplifiers.

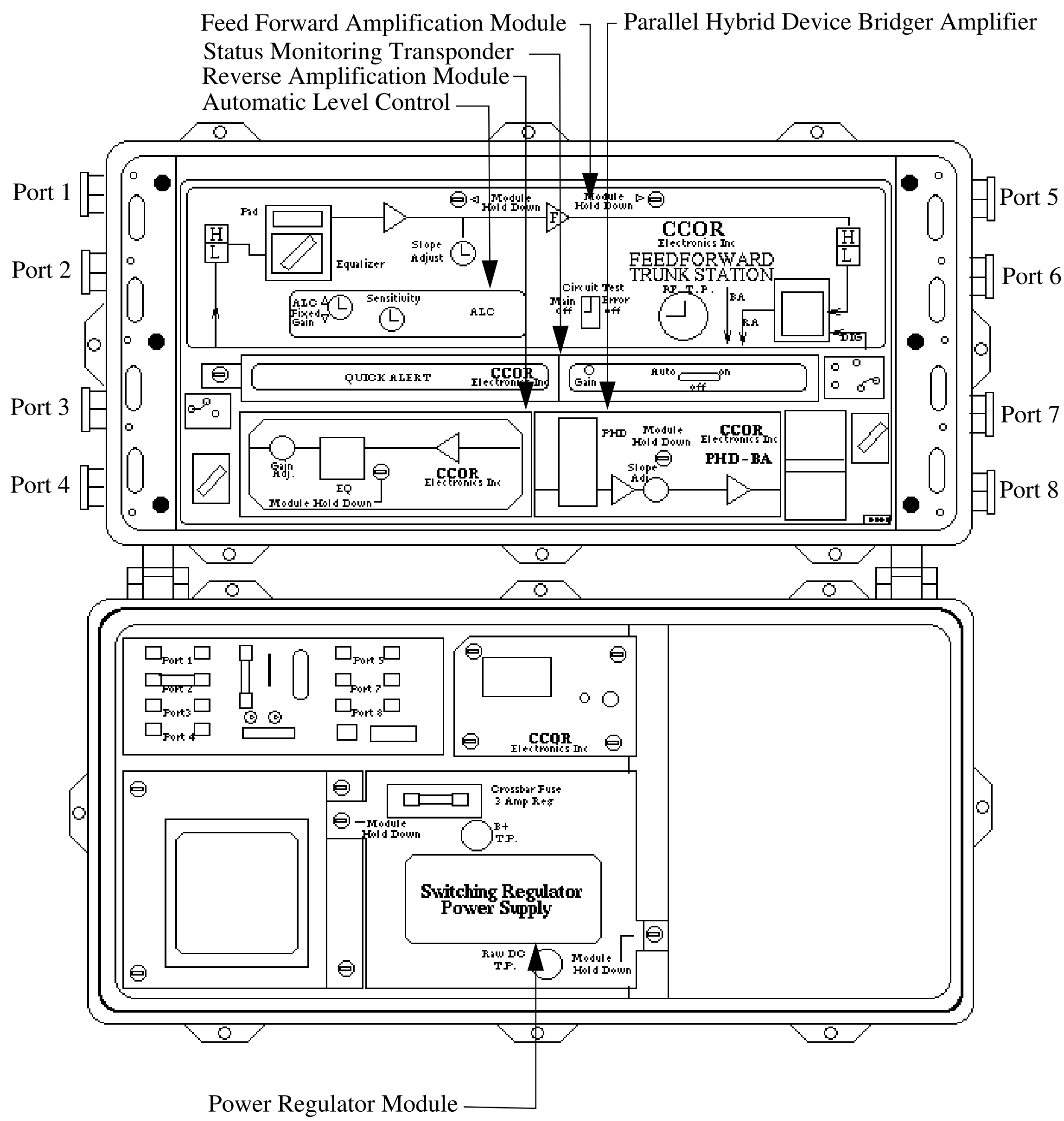


Figure 2.1: C-COR Cable Trunk Amplifier [12].

2.2.1 Forward Amplification

The main purpose of the cable amplifiers is to restore the losses incurred in transmitting the signals over the previous span of coax cable. This is the responsibility of the feed forward amplification module. Each amplifier operates at a specific central frequency, termed the pilot frequency, and the amplifier's preset amplification level refers only to its pilot frequency. In order to maintain the desired signal strength over the entire bandwidth, it is necessary to employ amplifiers with different pilot frequencies throughout the network. Typically, only three different pilot frequencies are needed to maintain the desired signal strength across the entire 500 MHz bandwidth. Table 2.1 lists the pilot frequencies employed in the Victoria plant and their associated signal levels in dBmV. The amplifiers are designed to maintain a flat, preset signal level across the entire 500 MHz bandwidth by only setting the amplification levels at these three pilot frequencies. The pilot frequency of the amplifier indicates the frequency at which the amplifier is trying to maintain its preset signal level. Each amplifier also produces ancillary amplification effects across the entire 500 MHz bandwidth. These ancillary effects are used to amplify the cable signals which are not located on one of the three pilot frequencies. This technique of leveling the signal levels in the network is illustrated in Figure 2.2.

Table 1.1: Victoria cable plant pilot frequencies and levels.

Pilot Name	Frequency (MHz)	Signal Level
High	499.25	39.0 dBmV
Medium	379.25	38.0 dBmV
Low	67.25	36.0 dBmV

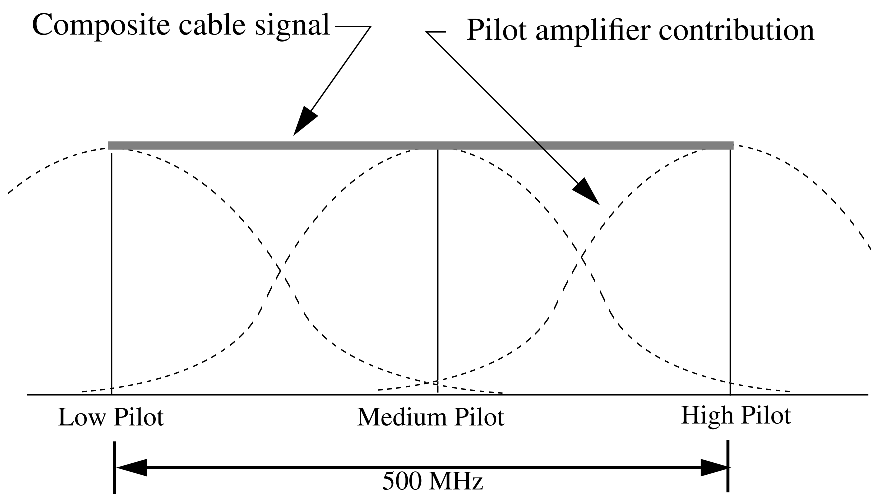


Figure 2.2: Network signal level amplification.

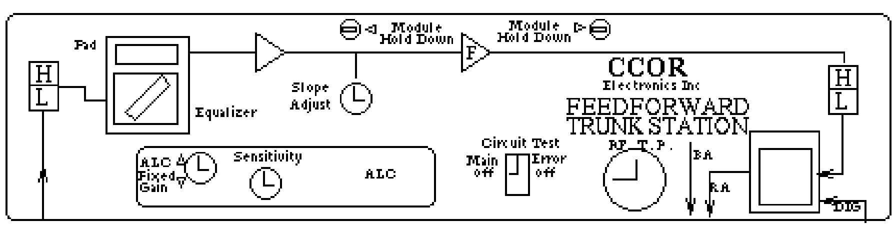


Figure 2.3: Feed Forward Amplification Module [12].

Because the signal losses in the coax cable span are frequency dependent the distance between amplifier of the same pilot type also varies. In the Victoria plant, one in every five amplifiers is a low pilot while the four amplifiers in between are all high pilots. The medium pilot amplifiers are actually remnants of older C-COR amplifier technology in which the network's operated at a lower maximum signal frequency. As such, these medium pilots do not occur at any pre-prescribed interval within the Victoria cable plant.

The losses incurred in a given cable span are dependent on the span's length and physical properties. For this reason, the feed forward amplifier must be tuned to its incoming coax span and a series of adjustable gain controls are present within the feed forward amplifier module to accomplish this task. Typically, these losses are in the range of 10 to 15 dBmV for most of the cable spans in the Victoria plant. The span will also have a relatively unique attenuation pattern across the frequency bandwidth of the cable signals. The feed forward module accommodates for this individual variance by having slots for equalizers and plug-in attenuators (PADs). When the technician installs a particular cable amplifier, equalizer and PAD modules are chosen with the particular values required to compensate for the given cable span's attenuation over the cable signal's bandwidth. These modules are then installed in the feed forward amplification module. This process is termed balancing the amplifier and must be done to each amp in turn starting with the amplifier located closest to the distribution centre. The net effect is to present a zero loss network at the leaf nodes of the trunk amplifier network. From the output of the feed forward amplification module the signal travels out of the amplifier through port 5 of Figure 2.1, designated the trunk signal output port, and, also, to the distribution section located within the amplifier. Port 1 is designated as the trunk signal input port.

2.2.2 Automatic Level Control (ALC)

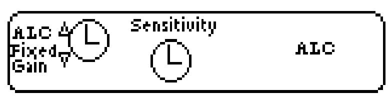


Figure 2.4: Automatic Level Control Circuit [12].

This is a special circuit actually located within the feed forward amplifier module. The purpose of this module is to compensate for previous amplification deficiencies which have occurred upstream. If, for some reason, the forward signal’s loss is more than can be accounted for by the preceding cable span, then the feed forward amplification module will be incapable of directly raising the signal up to the desired level. Therefore, the ALC activates and amplifies the signal up to the desired level, if possible. The ALC circuitry is limited to providing only a maximum signal amplification of between 3 and 5 dBmV. Any error that causes a loss in signal strength lower than the ALC can compensate for is only amplified up to the maximum ability of that particular ALC circuit. This circuit allows the network to continue functioning even though an amplifier's feed forward section is not working properly, or the input signal to the amplifier is below the level from which the feed forward amplifier can bring it back to the desire level on its own. This feature, though, can allow the network to operate even if it is at a marginal state, thereby, potentially masking a network fault.

2.2.3 Distribution

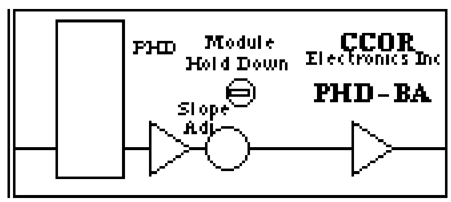


Figure 2.5: PHD BA Distribution Amplifier[12].

Once the signal strength and envelope have been restored by the feed forward amplification module, the signal is then passed on to the distribution module, termed the Parallel Hybrid Device Bridger Amplifier (PHD BA). This module is responsible for making the final correction to the cable signal before it is passed out of the four distribution ports on the amplifier, labeled ports 3,4,7, and 8 in Figure 2.1. This distribution module also has an insertion area PADs in order to achieve the desired signal envelope. These distribution legs may either feed another trunk amplifier or they may feed a local network of distribution amplifiers. Not all of these distribution ports necessarily need to be used in the same manner and some may not even be used in particular amplifiers. Reverse Amplification

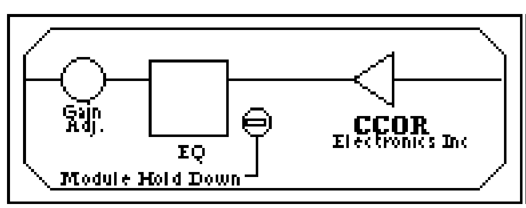


Figure 2.6: Reverse Amplification Module [12].

As mentioned previously, the C-COR amplifiers contain both a forward and reverse path. The reverse path occupies the frequencies from 5 to 35 MHz in the Victoria plant, and unlike the forward path multiple different types of amplifiers are not required to maintain a zero loss signal throughout the network. The purpose of the reverse path is to feed information acquired from the network or, alternatively, from sources connected to the network, back to the head end. Because of the relatively narrow bandwidth of these reverse signals, there is no need to have multiple different kinds of pilots centred on different pilot frequencies to maintain the desired signal levels throughout the network. The nature of the Victoria plant, though, does require that the reverse amplifier amplify the signals in such a way as to recover the signal losses that are going to be incurred in transmitting the signal upstream along the upcoming cable span. This makes maintaining the reverse path slightly more complicated in that the reverse amplification module of the amplifier downstream must be set properly to present the given amplifier with the desired reverse signal level. In the case of the Victoria plant this reverse signal level is set to 2 dBmV, where this 2 dBmV level is measured from the head end input to the given amplifier. With the exception of performing the signal measurements at the network's head end, the reverse signal path is balanced in a manner very similar to that done for the forward signals.

2.2.4 Power Supply

The electrical power for the amplifiers originates from separate power supplies, connected to a conventional power utility's power grid, located within the cable network. Each power supply is capable of providing power for up to four amplifiers and contains a separate battery backup for each of the amplifiers it supplies. This power signal is passed into the amplifier along port 2 and into the power regulator within the amplifier. The power regulator converts the quasi-square wave, 60 VAC signal generated by the power supply to the 24.0 DC voltage used internally by the amplifier.

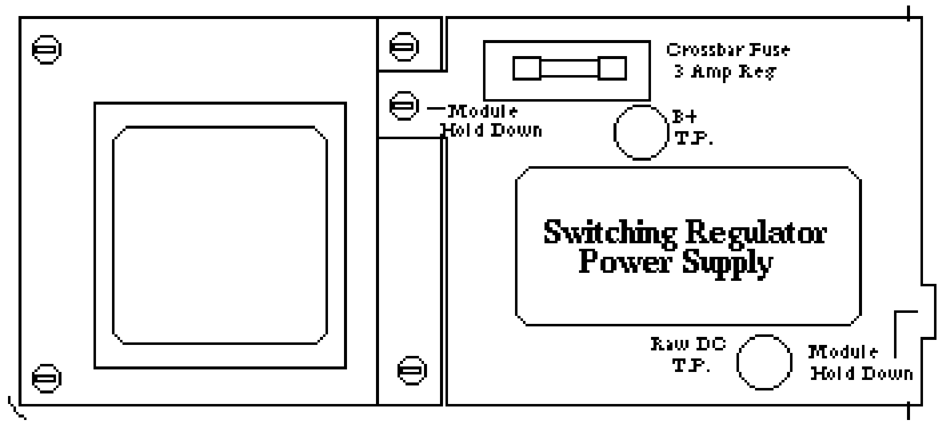


Figure 2.7: Power Regulation Module [12].

2.2.5 Failsafe Powered Housing

In the event of a complete failure, the C-COR amplifiers are designed to go into a bypass mode. In this mode, port 1 becomes directly connected to port 5 though the amplifier's metal case. Although the signal is not amplified when this mode is entered, it does allow the network to continue operation even in the event of a catastrophic failure in a given amplifier. In practice, over a series of amplifiers the signal slowly is picked up, and after about 4 or 5 amplifiers it is usually back to the 39.0 dBmV level specified for the Victoria plant. If the signal reached a significantly low level, it is possible that noise is also picked up with the signal and amplified to the 39.0 dBmV level. This event will cause problems to appear at the subscribers' end of the network, although it may not cause any additional errors to occur within the network itself.

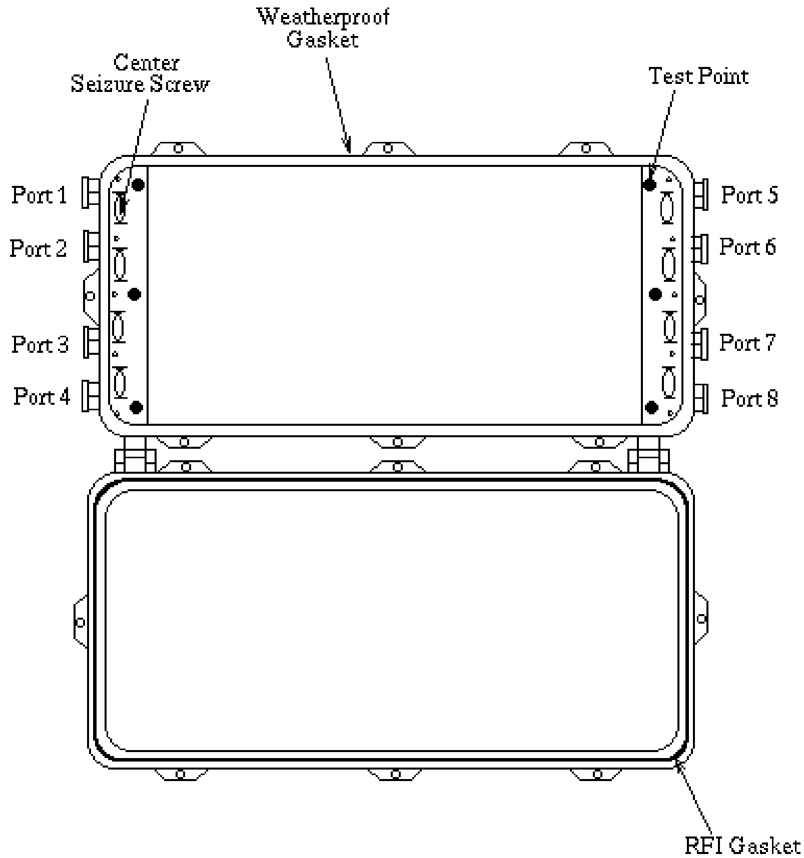


Figure 2.8: Failsafe Powered Housing [12].

2.2.6 Status Monitoring

If there is a desire to monitor the state of the amplifier, then an optional status monitoring transponder (SMT) may be installed in the amplifier housing. This module monitors specific signals within the amplifier and reports their values to the head end upon request using the reverse signal path. The signals that the status monitor observes are as follows:

1.Temperature.
2.B+ Voltage.
3.Raw DC Voltage.
4.Forward Pilot level.
5.Reverse Carrier Pilot level.
6.Current.

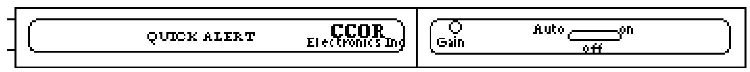


Figure 2.9: Status Monitoring Module[12].

The temperature is a measure of the temperature within the amplifier housing. By itself this measure is not very informative due to its wide variations, but it can indicate some sources of problems, such as a flooded manhole, when used in conjunction with the other fields. The raw DC and B+ voltages are the measured voltage supplied to the amplifier by an external power source before and after it has been internally regulated. These measurements can be used to indicate problems with either the external power supply or with the amplifier's internal power circuitry. The forward and reverse pilot levels are the measurements of the signal levels, in dBmV, of forward and reverse path signals centered at the pilot frequencies after their respective amplifications have been performed. As mentioned previously, the reverse path signal level is actually measured as the signal level from the amplifier in question to the head end as measured at the head end input. The current is the measurement of the total amount of current that is being drawn by the amplifier. Large variations in the value of this current may indicate internal amplifier faults. In addition to these six values, the SMT module can also supply additional information, such as the current state of the distribution module's Bridger switches. This information can be used to indicated potential faults in the network or sources of noise ingress. Each SMT modules is identified by a unique, programmable, binary address. In the Victoria plant, these addresses are referred to as the amplifiers SMT number.

The SMT modules also contain Bridger switches which when open allow reverse path signals to propagate through the SMT module into the reverse amplifier for transmission upstream. Normally, these Bridger switches are left closed, since downstream noise can also be introduced to the system along with reverse path signals when the switches are open. This is particularly true for trunk amplifiers at the network's boundary since they are the first line of protection from noise ingression by sources external to the network.

2.3 Trunk Cable Amplifier Network

The Rogers' Victoria cable plant currently consists of approximately 500 C-COR trunk amplifier connected in a single hub network. The size of the Victoria network permits this use of only a single distribution site, or hub. As the network expands, there may be the need to add additional hubs to the network in order to guarantee the quality of the cable signal. The addition of multiple hubs to a network does not in itself make the fault diagnosis task more difficult though since the hubs are independent and, therefore, can be diagnosed separately. The Victoria hub consists of a signal trunk cable amplifier located at the central distribution site which receives the original cable signals and transmits them to each of the three main trunk cascades. These trunk cascades each serve a particular section of Victoria, and are named accordingly to the street the main trunk proceeds along, specifically:

1. Esquimalt
2. Rutledge
3. Bethune

To allow for accurate mapping of this cable network, Rogers needed to name each of the individual amplifiers according to their physical network positions. Since the physical topology of the network is constantly being modified, this naming convention had to be robust enough to allow new amplifiers to be added to existing network branches. The naming scheme that was developed employed a 7 digit alpha-numeric number, which indicated the location of the amplifiers by identifying how many branches, starting from the hub, needed to be crossed in order to reach a given amplifier. For example the amplifier name R27BA03, indicates that the amplifier was the 3rd amplifier on the Ath leg of the Bth sub-leg of the 7th major leg of the 2nd cascade of the Rth hub in the plant (Figure 2.10). Alphabetical digits are used within this numbering scheme to allow a larger number of possible amplifier names without increasing the name past 7 digits. The sixth digit is used to indicate some special condition or use for the amplifier such as a bridger amplifier, a terminal amplifier, or a special service amplifier. For single hub networks, the 1st digit is redundant and may be used to provide an additional alphabetic cascade field as is done in the Victoria plant. This naming scheme is unique to the Victoria plant and, may or may not be employed by other Rogers' cable plants in Canada.

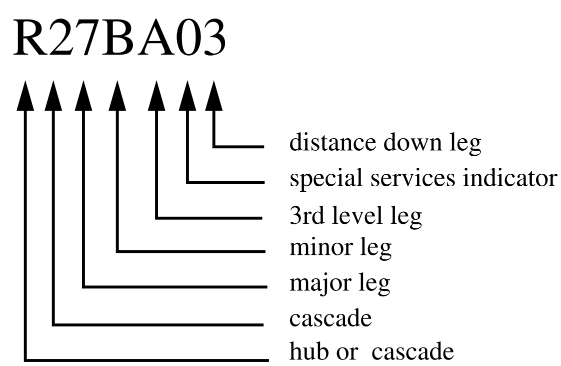


Figure 2.10: Rogers' amplifier naming scheme [15]

A symbolic map of the Victoria plant topology is maintained by Rogers and updated when network changes occur. Figure 2.11 is an example page of this map illustrating the use of the naming convention for part of the Esquimalt cascade. In addition to the naming information, the map also contains other information about the amplifiers regarding their function and physical location within Victoria. Figure 2.12 describes this additional information. The map symbol for the amplifiers consists of a small solid, black triangle, in the centre of a larger triangle representing the directions of the reverse and forward signal paths respectively. The example map page also indicates the external power supplies located on the network and their associated power supply numbers. Each power supply is capable of powering only a limited number of amplifiers and the boundaries of these power grids are indicated by the double lines across the signal path. Ideally, this paper map of the plant topology will match the actual topology identically but this typically not the case, so a constant revision process is required to keep the paper maps current with the latest network changes.

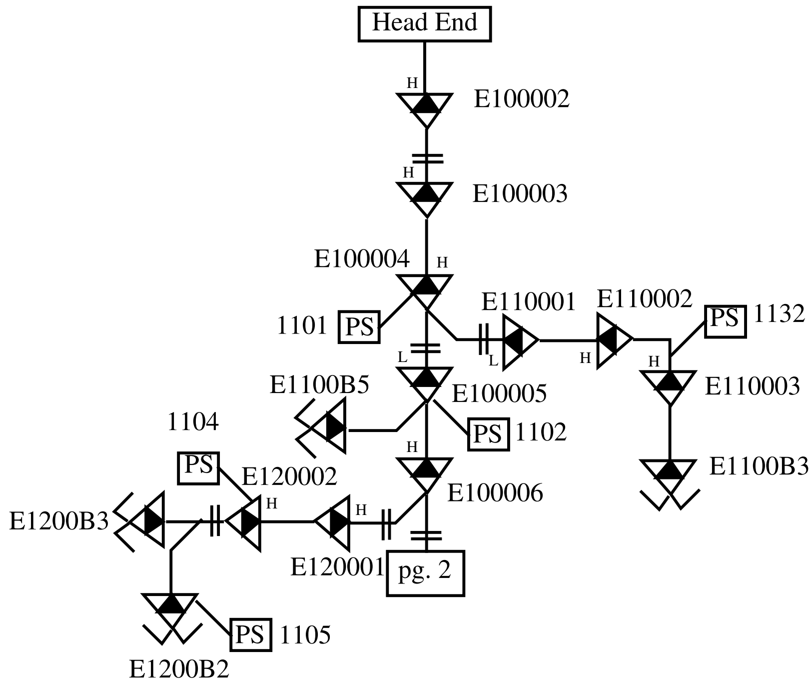


Figure 2.11: Example Victoria cable map page [15].

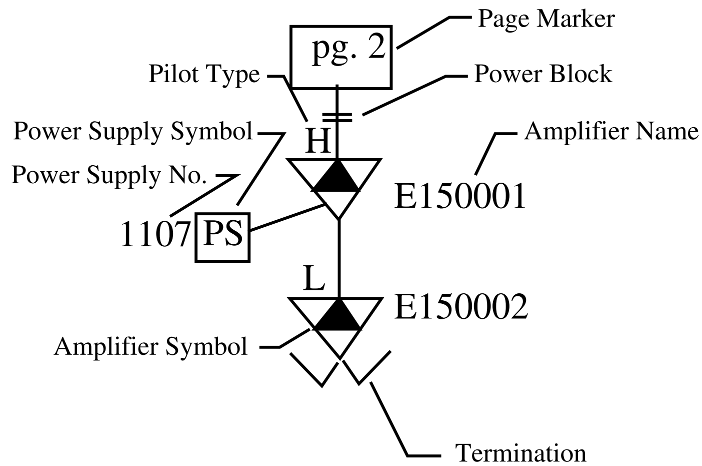


Figure 2.12: Amplifier symbol information [15].

2.4 Network Monitoring and Maintenance

Status monitoring software, produced by C-COR, is run on a personal computer located at the central distribution site and connected to the cable plant through a cable modem. This software continually polls each of the amplifiers with SMT modules in a cyclic order according to their SMT numbers. With the exception of a number of the terminal amplifiers, all the amplifiers within the Victoria cable plant are fitted with SMT modules and, hence, are monitored in this continual cycle. Since the SMT numbers are not related to the physical position of the amplifier in the network, the monitoring cycle appears as a random sampling of the network amplifiers. Figure 2.13 illustrates this polling cycle. Currently, all the amplifiers within the Victoria plant are polled within an interval of approximately 3 minutes. For a given network this interval is determined mainly by the physical number of to be polled. Typically for multi-hubbed networks, a status monitoring system must be located at each of the network hubs in order to provide reasonable polling intervals.

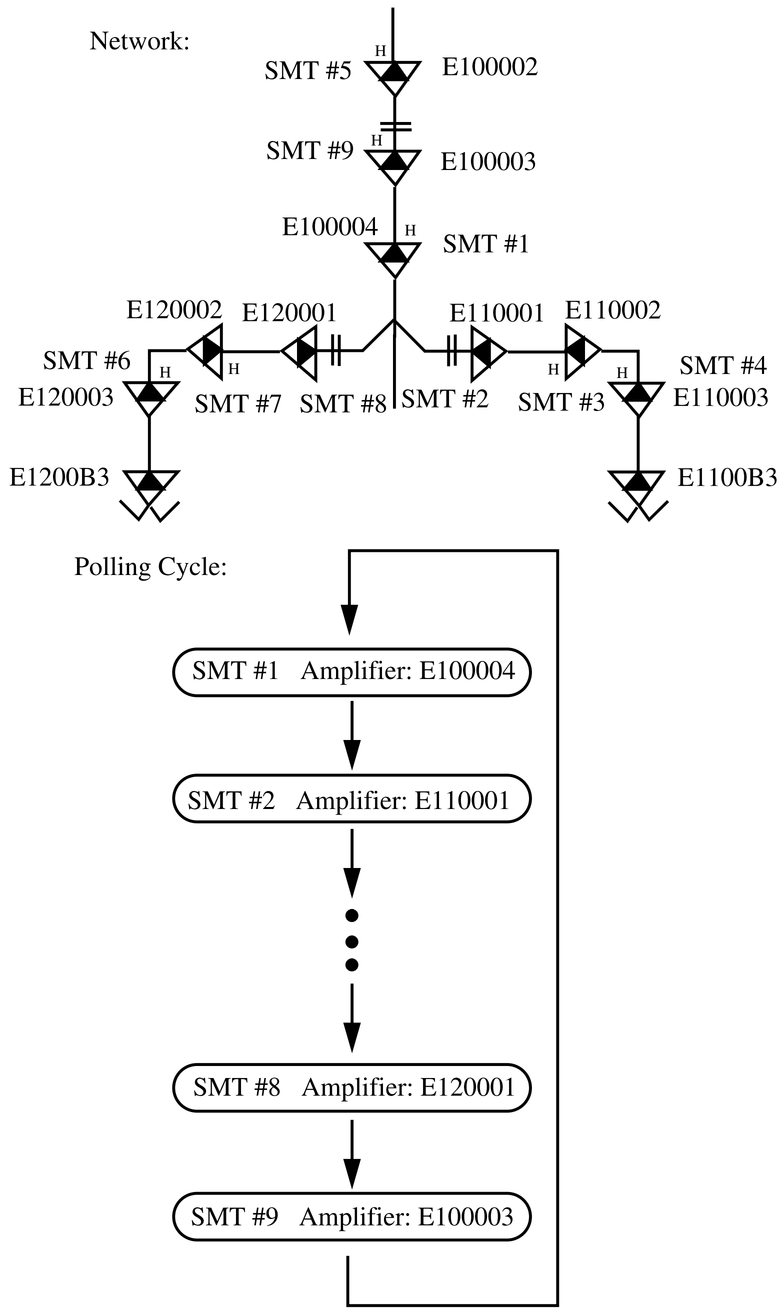


Figure 2.13: Amplifier polling cycle.

To poll a given amplifier, the SMT software must first ensure that there is an closed reverse path over which the amplifier can communicate. If such a path does not exist, due to the setting of the Bridger switches on intervening amplifiers, the software first closes the open Bridger switches along the path between the head end and the amplifier to be polled and then proceeds to poll the desired amplifier. Once the amplifier has responded to the SMT poll or the response time limit has been exceeded, the SMT software resets any of the modified Bridger switch settings.

After the SMT software receives the polled status from the given amplifier, it compares the returned values to the amplifier's set nominal values stored within the software's data base. This set contains two pre-set limits for each of the six measured fields. If a given measured value exceeds the first limit but is within the second limit a warning message is generated by the SMT software for that amplifier field. If the measured value for the given field exceeds the second limit then an alarm is generated by the software. If this alarm state exists for three consecutive polling cycles for the particular amplifier a failure message is then generated. Amplifiers which are within their first nominal limits are reported as being OK by the software. Figure 2.14 illustrates these progressively severe fault messages that are used in the Victoria plant for the forward pilot measurement. The SMT software only reports a warning, alarm, or failure for a given amplifier on the first polling cycle in which the amplifier is found to be outside its nominal range. If the amplifiers warning, alarm, or failure state does not change, the SMT software will not report the fault in subsequent polling cycles.

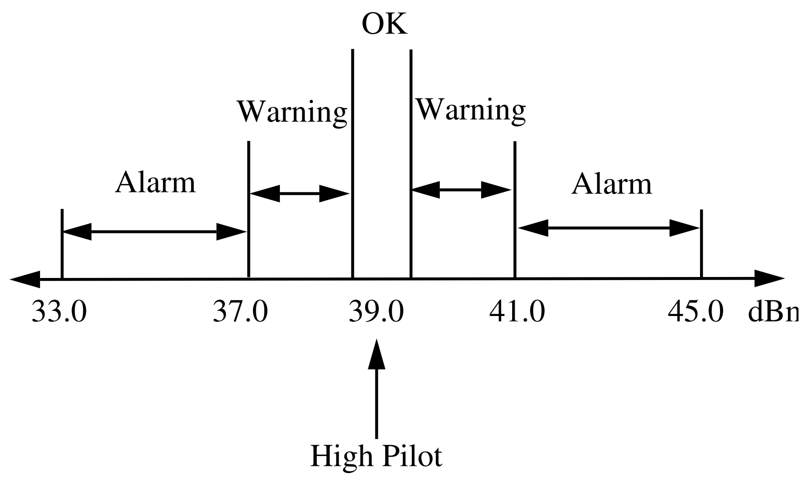


Figure 2.14: Example Fault message limits for forward pilot signal.

Since the amplifier nominal limits are statically set within the SMT software's data base, false error messages may be generated from the plant's natural drift. As the temperature varies across the amplifiers in the network, the various signal levels change slightly and the resulting signal amplification also vary. In some cases this natural drift of the signals is enough to exceed the given amplifier's nominal limits causing the SMT software to generate error messages even though the amplifier itself is working properly. In some cases, it is necessary to reset the nominal values for the amplifier in order to resolve these spurious error messages.

Once the SMT software determines that an amplifier is in a fault condition, the software displays the fault to the user in the upper right corner of the main screen in either yellow for warnings, red for alarms, or blue for faults, and specifies the amplifier's name, the fault type, and the fault message. The software also writes this error information both to a file on the computers disk and to a hard copy on an attached printer for future reference. An example section of this LOG file is shown in Figure 2.15. At this point the user may either allow the SMT software to proceed with its current polling cycle or the user can enter the software's monitor mode. In this mode, the software requests the user to provide the names of the amplifiers to be polled, either by entering directly the list of names or by entering name of a pre-defined network segment.

```
Clear logWGM 4/06 8:27
Log offWGM 4/06 8:27
R15A0T2     Warn:  Pilot RFval=35.8nom=39.0SYS 4/06 8:42
R15A0T2     Ok:    Pilot RFval=37.0nom=39.0SYS 4/06 8:45
R15A0T2     Warn:  Pilot RFval=35.8nom=39.0SYS 4/06 9:15
R15A0T2     Ok:    Pilot RFval=36.5nom=39.0SYS 4/06 9:18
B14000C     Ok:    Raw DCval=52.8nom=48.5SYS 4/06 9:24
R15A0T2     Warn:  Pilot RFval=35.8nom=39.0SYS 4/06 9:59
R15A0T2     Ok:    Pilot RFval=36.8nom=39.0SYS 4/0610:02
R15A0T2     Warn:  Pilot RFval=35.8nom=39.0SYS 4/0611:17
R15A0T2     Ok:    Pilot RFval=36.8nom=39.0SYS 4/0611:29
B140004     Warn:  Pilot RFval=35.8nom=39.0SYS 4/0611:35
B140004     Ok:    Pilot RFval=36.5nom=39.0SYS 4/0611:38
E14000B     Alarm: Unit lidSYS 4/0612:00
E14000B     Ok:    Unit lidSYS 4/0612:09
R15A0T2     Warn:  Pilot RFval=35.8nom=39.0SYS 4/0612:36
R15A0T2     Ok:    Pilot RFval=37.5nom=39.0SYS 4/0612:39
E1200T3     Alarm: Communicationval=OVERUNSYS 4/0612:57
Log onWGM 4/0612:57
Log offWGM 4/0613:01
```

Figure 2.15: Example section of LOG data file.

Once the software receives this list from the user, it proceeds to monitor only the specified amplifiers and presents the six field values, along with the amplifiers' names and lid status, to the user in the appropriate error colour codes. The lid status field indicates whether the amplifier lid is open or closed, and can be used to indicate whether the amplifier in currently being serviced by field technicians. Within this mode, the user may also graph a given amplifiers status in which case the software displays a bar graph, appropriately colour coded, showing the field values and their proximity to their associated nominal limits. It is important to note that when the SMT software is in this monitoring mode, the normal polling of the network is interrupted and does not proceed until the user exits from this mode. The SMT software also stores all the values returned by the amplifiers in the network during the polling cycle along with the amplifiers' names and the date and time the data was received in a set of QA dump files on the computer, an example section of which is shown in Figure 2.16. Because of the volume of information stored within these files, they are normally purged from the status monitoring system.

```
03301359RUT HE      SMT R=  2.0 P= 39.0 D=----  B=24.0 RD=56.1 C=1143.3 T=  1.0
03301359ESQ HE      SMT R=  2.0 P= 38.8 D=----  B=23.6 RD=57.8 C=1180.2 T=  2.0
03301359BET HE      SMT R=  2.0 P= 39.0 D=----  B=23.7 RD=55.3 C=1088.0 T=  1.0
03301359B160003     SMT R=  2.0 P= 38.8 D=----  B=23.8 RD=57.4 C=1014.2 T=  6.0
03301359E100002     SMT R=  0.0 P= 38.8 D=----  B=24.0 RD=49.4 C=1124.8 T= 20.0
03301359E100003     SMT R=  2.0 P= 39.3 D=----  B=23.9 RD=53.6 C=1088.0 T= 12.0
03301359E100004     SMT R=  1.0 P= 39.0 D=----  B=23.9 RD=54.9 C=1143.3 T=  3.0
03301359E100005     SMT R=  1.0 P= 36.0 D=----  B=23.8 RD=55.3 C=1124.8 T=  0.0
03301359E100006     SMT R=  1.0 P= 38.8 D=----  B=23.7 RD=48.5 C=1088.0 T= -5.0
03301359E100007     SMT R=  1.0 P= 39.3 D=----  B=23.9 RD=56.1 C=1124.8 T= -3.0
03301359E100008     SMT R=  1.0 P= 38.0 D=----  B=23.8 RD=49.4 C=1088.0 T=  6.0
03301359E100009     SMT R=  0.0 P= 40.3 D=----  B=23.9 RD=48.1 C=1124.8 T=-14.0
03301359E10000A     SMT R=  0.0 P= 39.3 D=----  B=24.1 RD=54.4 C=1124.8 T=  5.0
03301359E170001     SMT R=  1.0 P= 38.0 D=----  B=24.1 RD=54.0 C=1198.6 T=  3.0
03301359E10000B     SMT R=  2.0 P= 37.3 D=----  B=23.9 RD=53.2 C=1106.4 T=  9.0
```

Figure 2.16: Example section of QA dump data file.

Typically, the SMT software system is mainly used only to help track down faults once repair crews have been dispatched to the general area suspected to be in failure. The process of locating the general area of the fault is commonly done either through subscriber complaints received by Rogers' operators or by experienced personnel using the SMT software, in conjunction with the plant maps, to determine the last amplifier in the given network line that is functioning properly. From this point, the field repair crews either rely on their own intuition and experience to determine the cause of the fault and institute repairs, or the intuition and experience of the network experts. If the initially suspected problem is not the cause of the fault, then the repair process, typically, proceeds by a trial and error approach until the fault is located and fixed. Because isolating the specific cause of the observed fault is difficult, if not impossible, in the current maintenance process employed by Rogers, when an amplifier is found to be the cause of a fault it is replaced by a spare amplifier that is known to be good. No attempt is made to isolate and replace faulty amplifier modules in the field. Instead the faulty amplifier is brought back to the central distribution site for repairs. This means that each of the repair crews must carry a given number of spare amplifiers and must spend the time re-balancing the amplifier line whenever a fault is located that requires the amplifier to be swapped.

The SMT software can generate fault messages for amplifiers that are operating correctly. This can occur either because of plant drift, as mentioned previously, or because an actual fault occurred within the network but its duration was within the network's polling cycle. For example, if an external power supply momentarily fails because of a Hydro outage, the SMT software will report fault messages for the amplifiers connected to its power grid during the polling cycle. If Hydro returns during the given polling cycle, but after the particular amplifiers have been polled, these amplifiers will be reporting error conditions even though they are functioning properly. On the subsequent cycle, the given amplifiers will report an "OK" status to the SMT software indicating the fault condition no longer exists. To alleviate this problem of the SMT software indicating non-existent faults, Rogers' personnel first determine whether the given network fault is producing downstream and upstream effects before repair crews are dispatched. Any faults which do not produce these effects are deemed to be spurious faults and ignored except in the case where they occur repetitively, in which case they indicate problems within the given amplifier that is reporting the fault.

2.5 Failure Modalities

Cable trunk amplifier networks are subject to particular types of network faults. The purpose of this section is to present some of these common network faults and to illustrate the behaviour that cable networks exhibit during these fault modalities. Some specific properties of cable networks will also be illustrated by this discussion. The failure modalities that will be presented in this section are: power supply faults, internal amplifier faults, and amplifier interconnection faults. These failure modalities represent the most common causes of network failures since they cover the set of failures associated with any of the network's three principle subsections, namely the amplifiers, the power supplies, and the cable spans. Other failure modalities may exist but they are currently unknown and are assumed to occur relatively rarely. The diagnostic tool, therefore, presently only addresses these three major fault modalities.

These are the only failure modalities that have currently been isolated from the LOG and QA dump data files collected during the January to May 1992 time period. These modalities represent the most common types of network failures and, hence, were the modalities addressed by the diagnostic tool that was created.

2.5.1 Power Supply Fault

Each of the external power supplies contains battery back ups to be used in the case of a Hydro outage. These batteries will supply power to the amplifiers connected to the given power grid for approximately 1 hour. Once the batteries are drained, the SMT software will then start to observe error conditions in these amplifiers. Once power is cut to the trunk amplifiers, they enter their bypass mode, therefore, the cable signals do proceed to the downstream amplifiers. Since no amplification has taken place for these signals, they reach the downstream amplifiers severely attenuated by the two cable spans and by the bypassing amplifier housing. The attenuation may be severe enough to place the cable signals within the network's background noise levels, in which case all subscribers who receive cable signals from points below the power supply failure will be effected.

Typically, these downstream signals will be attenuated by at least 20 dBmV which is far below the amplification abilities of a single amplifier since the amplifiers are typically set for around a 10 dBmV gain and their ALC modules can only at best contribute another 5 dBmV of gain. Therefore, several amplifiers in each of the downstream lines for all the amplifiers on the failed power grid will also be reporting faults. Typical power supply failures in the Victoria plant, result in at least 4 downstream amplifiers being effected. The exact number of amplifiers effected is dependant on the actual cable span lengths involved and the ability of each of the amplifier’s ALC modules to restore the cable signal. If the signal level reaches a sufficiently low level, the downstream amplifiers may not be able to restore the cable signal back to the nominal levels. In this case, the SMT's polling requests also will not reach the downstream amplifiers and these amplifiers will be reported as having no reply to the SMT's requests. Because the external power supplies do not contain SMTs, it is usually difficult to determine whether a given failure of this type is due to a Hydro outage or to a fault internal to the specific power supply. In the case of a large Hydro outage, though, this differentiation is simpler due to the large number of effected amplifiers. To address this fault isolation problem, Rogers is planning to implement power supply monitoring within the coming year.

2.5.2 Internal Amplifier Fault

Unlike power supply faults, internal amplifier faults may or may not cause associated faults in neighboring amplifiers. If the fault is not severe then only the particular amplifier will be reporting a fault condition. If, on the other hand, the fault is severe enough to cause the incoming cable signals to pass through unamplified or not at all, then the fault condition will produce downstream effects very similar to those produced by a power supply failure. The only difference will be that the effects will only be limited to one amplifier's downstream path not a group of downstream paths. Any situation between these two extremes is also possible, and the actual number and type of downstream faults reported is dependent on the actual fault that occurred within the given amplifier. As mentioned previously when these types of faults occur, repair crews swap out the defective amplifier instead of attempting to isolate the particular module that has failed.

2.5.3 Amplifier Interconnection Fault

This class of faults involves problems that occur due to faults within the cable spans interconnecting the amplifiers. These faults can range from physical line breaks to bad splices between cable sections within the span, or physical defects within the cable. Typically with this class of faults, a number of downstream amplifiers will be affected. In the case of a line break, all the amplifiers below the break will be reported as having communication faults by the SMT software. Bad connections and physical defects within the cable spans usually result in an attenuation of the high frequency cable signal. Therefore, it is possible that such a fault may occur in a span directly preceding a low pilot amplifier and an error condition will not be reported by the SMT software for the low pilot amplifier but will be reported for high pilot amplifiers further downstream. The low pilot will not report a fault condition because its pilot measurement is performed only for the part of the cable signal about the low pilot centre frequency which is not affected by the cable problem. Although the frequency of this type of problem is relatively low, it does indicate that under specific conditions amplifiers may report an OK status even though the trunk line they are located on is in a fault condition.

2.6 Chapter Summary

This chapter presented a description of the current Rogers' Victoria cable plant and an introduction to the functioning of the trunk amplifiers which form the key elements of this network. A description of C-COR's status monitoring system was presented along with a brief description of the current diagnostic procedures employed by Rogers' staff. The chapter concluded by discussing three of the main failure modalities of this type of network and the observable amplifier faults that result from each of these failure types. This section also illustrated how faults resulting from different causes can produce very similar network effects.

Chapter 3:

Network Modeling

3.1 Introduction

The prototype network diagnostic tool was designed according to the principles outlined in Chapter 1. These principles state that modeling of the network in terms of its structure, function, and behaviour are essential in order to produce a sufficiently accurate and robust diagnostic tool. Since the Rogers' trunk amplifier network is a homogeneous network, only its structure and behaviour need to be known in order to completely model the network. The first step in developing the prototype diagnostic tool was to create the software tools to allow for this structural and behavioural modeling. This Chapter will discuss the development of these tools and the design decisions that were made in their production. Because this work is a prototype of the final project goal, namely the creation of the complete, stand alone, real-time, expert system based network analyzer, the design decisions that were made were based on both current needs and on future software requirements. Since the overall project will consist of a relatively large number of integrated software tools, care was also taken to ensure that software engineering principles, such as modularity and information hiding, were employed in the tools' development. C was chosen as the development language because of its portability and the ease of integrating it with existing programs.

In support of this work, Rogers has installed a cable modem to one of the Sun workstations at the University of Victoria. This link currently provides a 19.6 Kbaud data path over which the QA dump and LOG files for the entire Victoria cable plant are transmitted daily. These files when combined, fully represent the behaviour of the entire network over the applicable time period. Where applicable, the software tools were written in such a manner that the majority of the code could be utilized within future upgraded versions of the system.

3.2 Structural Modeling

Rogers documents the current physical structure of its Victoria cable plant through the use of an electronic data base located on an MS-DOS computer. Figure 3.1 illustrates the format of this network description. In order to build an automated diagnostic tool, this network topology needed to be represented in a form that is easily utilized by the diagnostic tool. This representation was accomplished through the development of a set of software tools collectively termed the topology server. This software represented the network in an hierarchical, object-oriented structure illustrated in Figure 3.2.

STREET	BINADDR	AMP_NUM	DESCR	AMP_LET	MODADDR	RET_MOD	PWRSUPPLY	SHEET	PREV	FOLLOW1	FOLLOW2	FOLLOW3
Adelaide@2835	NIL	1471	2969 2C		E1600B4	0	1109	081	1470			
Admirals&hallowel	351	1990	419RH 2C		E190007	0	1122	66	1260			
Admirals@arundel	94	1481	409RH 2C		E160006	0	1111	98	1480	1780		
Admirals@Astle	550	1330	409RH 2C		E1CB001	0	1130	42	1311	1331		
Admirals@Cowper	354	1480	409RH 2C		E160005	0	1110	98	1750	1781	1770	
Admirals@esson	NIL	1780	2969 2C		E1600B6	0	1111	98	1481			
Admirals@garry	NIL	1770	2969 2C		E1600B5	0	1110	80	1480			
Admirals@isbister	330	1240	419RH 2C		E190004	0	1121	54	1230	1250		
Admirals@modeste	341	1260	409RH 2C		E190006	0	1122	066	1250	1990		
Admirals@naden	NIL	1331	2969 2C		E1CB0B2	0	1130	042	1330			
Admirals@parkland	359	1250	419RH 2C		E190005	0	1122	54	1240	1260		
Alder@tolmie	5	1000	519RL 2C		E100002	0	1000	070	0010	1010		
Alderley@ 5365	580	5260	519RL 2C		B14000Q	0	1521	186	5250	5270		

Figure 3.1: Rogers' Victoria Cable Plant Description.

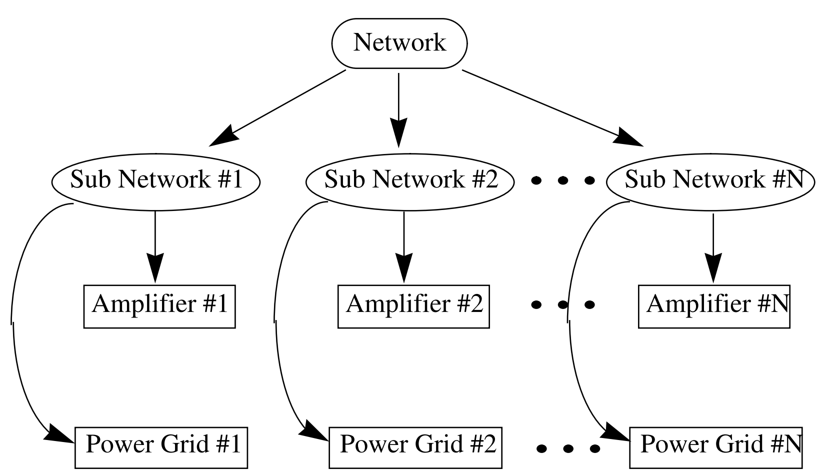


Figure 3.2: Topology Server Class Hierarchy.

As this Figure shows there are four main classes of network objects, namely:

1.Network.
2.Sub Network.
3.Power Grid
4.Amplifier

The following sections will describe each of these classes and their respective software modules in detail. Since object-oriented constructs are not defined within the C language, these object classes were implemented in the form of C structure definitions and the links between objects were implemented using linked lists created through the use of C's pointer construct. For coding efficiency, all the linked lists within the topology server operate on Sub-Network objects and utilize the same set of generic linked list routines. These routines provide the basic functionality needed to create and destroy lists as well as to add and delete objects within the list. The code within the topology server does not rely on the particular type of network elements and, therefore, could be easily used to represent other types of networks.

3.2.1 Network Module

The network class is the top level in the hierarchy and it represents the network as a whole. Table 3.1 illustrates this class and the fields within it. The trunk list field contains a pointer to the head of a list of the network hub's main branches. The sub network list field contains a pointer to the head end of a list of all the Sub-Network objects within the given network. Likewise, the power grid list and amplifier list fields, respectively, contain pointers to the head end of the list of all the power grids and all amplifiers in the network. The entire structure of the network is contained within these three doubly linked lists. The final two fields in the network class are used to specify a text string indicating the coverage area of the network and a pointer field to allow for future expansion of this network class. In the future, for multiple hub networks a class above the network class may need to be created to maintain a linked list of the various network objects that exist in the complete network. In this case the network level objects would represent each of the given network's hubs. Since the Victoria cable plant is only composed of a single hub, the hub’s head amplifier is not included in the description of the Victoria cable plant.

Table 3.1: Network class structural and routines.

Type	Routine
pointer	Trunk list head
pointer	Sub Network list head
pointer	Amplifier list head
pointer	Power Grid list head
string	Network area string
pointer	Extensions

.

Table 3.2: High level network class routines.

Routine	Description
read_network	Loads network from network description file creating all of the associated linked lists and object interconnections.
write_network	Writes out computer image of network in the network description file format.

Access routines, in the form of get and set functions, were written to modify and retrieve information from each of the network fields. In addition, routines to read in and write out the cable plant description file were also created. The read network routine creates an image of the cable network within the computer memory for use in diagnostic processing. The write network routine takes this computer image and writes it back to a file in the original network description format. This routine was created for testing purposes since the newly created network description can be compared to the original description file. If the two files are identical, then that provides fairly strong evidence that the topology server software is functioning properly. Due to the difficulty in verifying C pointer assignments for large data structures, this file comparison provides one of the few methods of verifying the topology server software. Routines were also included to perform searches along each of these lists.

3.2.2 Sub Network Module

The sub network class is used to store the information about how the various network components are interconnected. Table 3.3 illustrates this network class and its various fields. The amplifier field contains a pointer to the amplifier object, which is located in the list pointed to by the network object's amplifier list field. This field can be used to uniquely identify the amplifier to which the sub network information, contained in the remainder of the fields, applies. The power grid field is similar to the amplifier field except that it contains a pointer to the power grid object associated with the given amplifier. This object is located in the power grid list pointed to by the power grid list field of the network object.

Table 3.3: Sub network class structure and fields.

Type	Routine
pointer	Amplifier
pointer	Power Grid
pointer	Branches list head
pointer	Parent
pointer	Extensions

.

Table 3.4: High level sub network class routines

Routines	Description
add_branch_subnet	Adds a sub network object to the given sub network's branch list.
find_amp_subnet	Finds the subnetwork object associated with the given amplifier name.

The branches field contains a pointer to a list of the downstream amplifiers, which are themselves sub network objects. Since the connections to the downstream amplifiers are implemented in terms of elements in a linked list, there are no restrictions as to the degree of branching that is allowed for any given amplifier. One problem with this approach, though, is that there is also no inherent means of identifying which of the amplifier's ports feed a particular branch. Currently, this limitation does not affect the performance of the diagnostic tool since it is assumed that every port is affected equally by a given fault. If need be, this difficulty can be overcome in the future since the naming scheme employed by Rogers allows for the independent generation of the name of the amplifier supplied by a given port. The branch list can then be searched, and the sub network corresponding to this amplifier found. This port information is available in the network description file, although, it is currently unused. To aid in searching along the network structure, a previous field is also included in the sub network class. This field contains a pointer to the sub network which is the parent, or upstream amplifier, of the given sub network. Finally, an extension field is also included for future expansion of the sub network class.

The sub network objects can be viewed as representing the individual amplifiers within the given cable network. Information associated with the amplifiers, such as their names and power grid numbers, is stored within the lower level objects, thereby separating the network topology from this additional information. The entire physical structure of the network is stored solely within the sub network linked list pointed to by the network object's sub network list field. As in the case of the network module, get and set routines were written to modify the various sub network class fields.

3.2.3 Power Grid Module

This class is at the lowest level in the topology hierarchy and identifies information about particular power grids. Specifically this information is the power grid identity number, stored in the first field, and a list of amplifiers which are powered by the given power grid, stored in the second field as a list of sub network objects. The remaining two fields in this class are used in the maintenance of the power grid linked list which is pointed to by the power grid list field in the network class. These fields are not accessed directly by the diagnostic tool. As in the previous modules, get and set routines were written to modify each of these fields. In addition a search routine was created which would, given a power grid identity number, search the power grid list and return the names of all the amplifiers supplied from that given power grid. In the near future, Rogers is considering adding power grid status monitoring to the Victoria cable network in which case this class will be expanded to store any additional information about the power grids that is needed. Because all the information specific to a given power grid is stored within a single power grid object and access is achieved through the use of C pointers, there is no duplication of information storage within the topology server.

Table 3.5: Power grid class structure and fields.

Type	Routines
integer	Power grid identifier
pointer	Sub network objects list head
pointer	next
pointer	previous

.

Table 3.6: High level power grid class routines.

Routines	Description
find_power_grid_id_power_grid	Finds the power grid object associated with the given power grid identifier.
find_subnet_power_grid	Finds the power grid objects associated with the specified sub network object.
create_and_set_all_power_grid	Creates a new power grid object, setting all the fields as specified and adding it to the end of the power grid list.

3.2.4 Amplifier Module

The amplifier class is very similar to the power grid class with the exception that it stores information concerning particular amplifiers. Table 3.7 illustrates the various types of information that is stored. The name field stores a text string corresponding to the amplifier name as generated by Rogers' Victoria naming scheme. The location field stores a text string corresponding to the physical location of the amplifier in terms of the nearest street intersection. The type field contains a text string identifying the actual model name of the amplifier. The SMT number field, as the name implies, contains the SMT number associated with the amplifier. As mentioned in the previous chapter, this number also identifies the amplifier's position in the SMT polling cycle. The subs field contains an integer representing the number of distribution amplifiers that are supplied by the given trunk amplifier. This number can be used to give an indication of how many subscribers are affected by a given fault. The pilot field stores a character string indicating whether the amplifier is a low, medium, or high pilot. The termination field also contains an integer number which indicates if the given amplifier is located at a boundary. There are two types of boundary conditions that the diagnostic tool needs to be aware of. These are that the amplifier does not feed any subsequent downstream amplifiers, or, if it does, that these downstream amplifiers do not have status monitoring transponders. Because of how the diagnostic tool performs its reasoning, it is important that the topology server be able to distinguish between these two different types of boundary conditions. The page marker field is currently not used, but was intended to provide the ability to produce paper maps identical to the current maps employed by Rogers' personnel as a means of verifying the internal network representation. Because of time limitations and inaccuracies within Rogers' paper maps, this map generation ability has not been added to the topology server. As in the power grid class, the two final fields are used in the internal generation of the amplifier linked list, the head of which is pointed to by the amplifier list field in the network object. Like the power grid information, there is no duplication of the amplifier information within the topology server. As in all the other modules, the appropriate get and set routines were created for modifying the values of the various amplifier class fields.

.

Table 3.7: Amplifier class structure and fields.

Type	Routines
string	Name
string	Location
string	Type
integer	SMT number
integer	Number of subs
string	Pilot type
integer	Type of termination
integer	Map page marker
pointer	next
pointer	previous

Table 3.8: High level amplifier class routines.

Routines	Description
find_amp_amp_list	Finds the amplifier object with the given amplifier name.
create_and_set_all_amp_list	Creates a new amplifier object, setting all of the fields as specified and adding it to the end of the amplifier list.

3.3 Behavioural Modeling

A set of routines to access the information stored within both the LOG and QA dump data files for particular amplifier at particular times needed to be created in order to allow the network's behaviour to be extracted for these files. This combined information obtained through these routines can then be used to build an accurate behavioural model of the desired portion of the network during the specified time period. Because of the relatively large size of each of the collected QA dump data files, on the order of 16 Mbytes, a special set of routines had to be created to manage the accesses to and storage of this information[1]. Currently, the collection of LOG and QA data files extends from December 1, 1991 to the present and, therefore, the network behaviour can be extracted for any period within this collection window. As the data link is improved and the SMT software is upgraded to allow for real-time data request for given amplifiers, these two sets of data access routines will have to be replaced. Because the diagnostic tool makes generic requests to these routines, specifying just the desired amplifier and time period, this change to a real-time system should be relatively simple.

3.4 Chapter Summary

This chapter presented a discussion illustrating how the structural and behaviour modeling of the Victoria cable plant was achieved. An explanation of the design decisions made during the creation of the software tool necessary to perform this modeling was also presented. It is important to note that the accurate structural modeling of the cable network is a key requirement of the diagnostic tool. If the structural modeling of the network is inaccurate, then the diagnostic tool will not be able to perform adequately. Currently, Rogers does not have a specific policy for the maintenance of their network database and, during the course of implementing the topology server, a relatively large number of errors within the network description file were identified. The future accuracy of the description file will play a key role in efficiency of the final network analyzer in terms of the number of faults it is capable of diagnosing accurately. In the future, time permitting, work may be done on creating a small program capable of identifying some of the most obvious faults within the description file, such as an amplifier being connected to more than one power grid, to address part of this problem.

1. These routines were created by Mr. Andrew Watkins and are mentioned merely for completeness.

Chapter 4:

Fault Processing

4.1 Introduction

The next step in the development of the prototype diagnostic tool was to create the diagnostic software itself. As illustrated by Figure 4.1, this software was implemented in two stages: the cluster server, and the cluster diagnostic system. A single network fault will cause multiple amplifiers to report errors to the SMT software because of signal degradation. It is, therefore, necessary to be able to determine how the faults reported by the SMT software are interrelated. The term *fault cluster* is used within this work to refer to a group of amplifiers which are generating interrelated SMT error messages because they are affected by a common network fault. The clustering function is performed by the cluster server section of the diagnostic tool. Once the reported faults have been processed to produce a set of amplifiers effected by a given fault, then this cluster is passed on to the cluster diagnostic section to determine the actual cause of the fault occurring within the cluster. This cluster diagnostic system is implemented as a knowledge and rule base within a commercial expert system shell. This commercial shell was used to expedite the development of this cluster diagnostic system.

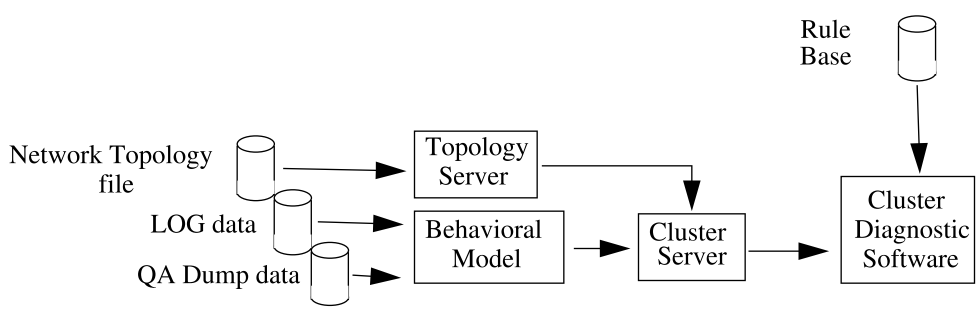


Figure 4.1: Overview of cable amplifier network diagnostic tool.

This chapter begins by presenting a general definition for these fault clusters and illustrates the spatial and temporal changes that occur to the cluster during its life cycle. This section will also discuss how the use of this cluster model allows for the development of a generic diagnostic tool and how multiple, co-existing network faults impose limits on this tool. The cluster server, the first stage of the diagnostic tool, will then be presented. This presentation will be followed by a discussion of how the cluster diagnostic stage of the tool was implemented. The discussions of these two stages will focus particularly on their implementation, their interconnection, and the associated design issues involved in their development. In the case of the cluster server, the discussion will also illustrate how the general cluster definition was modified in the context of developing a prototype diagnostic tool and the resulting implications.

It is important to note, that this work represents only the first stage in the evolution of the complete, real-time diagnostic tool. As such, this prototype tool only diagnoses a subset of all possible network faults, although this set does represent the majority of the most common types of faults. This diagnostic system also only deals with network level faults. Faults below the network level, such as a fault within a given amplifier module, are not diagnosed by this system. These lower level faults, though, could be included in the diagnoses by adding additional knowledge bases to the current diagnostic tool, as illustrated by the radio network diagnostic tool presented in Chapter 1. Associated research is currently being pursued in developing a knowledge acquisition tool, called FLOWTOOL, which will aid in the creation of these additional knowledge bases. Knowledge bases at the amplifier diagnostic level have been created using this tool, but these knowledge bases have not yet been incorporated into the diagnostic tool and, hence, will not be discussed within this work.

4.2 Fault Clusters

The general definition of a *fault cluster* is the set of network elements that are effected by a given network fault to such an extent that they themselves report an error condition. For example, Figure 4.2 illustrates a section of the amplifier network and the fault messages associated with a communication problem between amplifiers E10000F and E10000G. The extent of the fault cluster is indicated by the line encompassing all of the affected amplifiers. The amplifiers at the edges of this cluster are termed the *boundary* amplifiers, and are the last amplifiers that are significantly affected by the given fault. Amplifiers further downstream may see some effects of the given fault, but these effects will not be significant enough, by themselves, to cause any error messages.

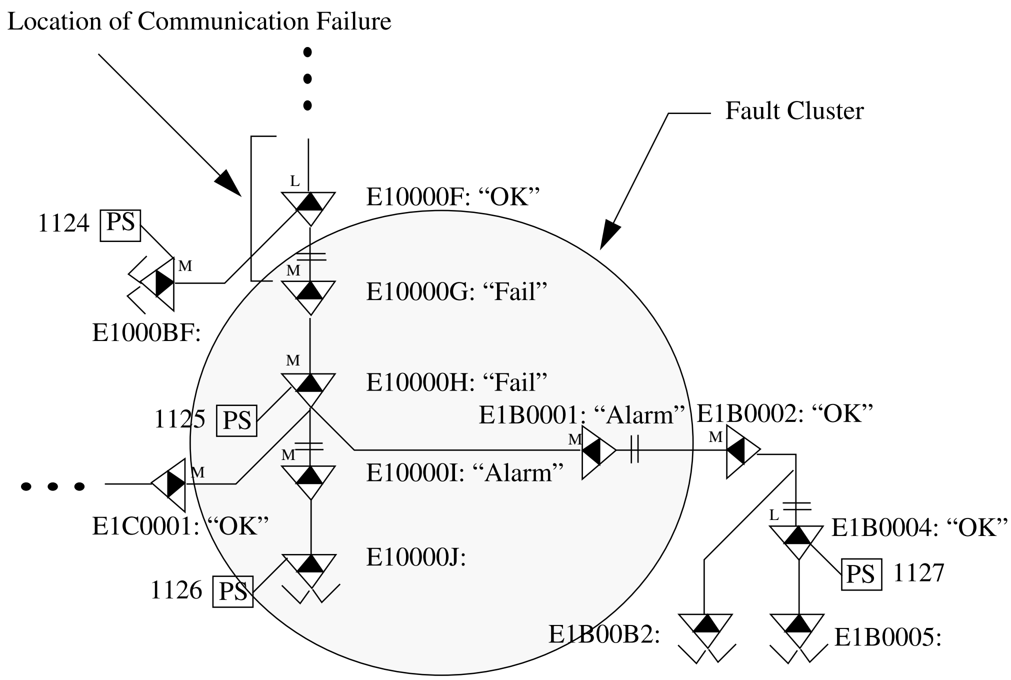


Figure 4.2: Example fault cluster.

In general, a cluster will change both temporally and spatially during the life cycle of the given fault. For example, as illustrated in Figure 4.3, a fault cluster may begin with a single amplifier at time t_0. As time progresses the cluster will grow to include more amplifiers due to the propagation of the deteriorated signal levels to amplifiers further from the site of the fault. Eventually, the maximum number of affected amplifiers will be reached and the cluster will cease to grow, assuming no additional faults within the cluster occur. Once this maximum cluster size is reached, the cluster elements will not generally remain constant since the amplifier on either side of the cluster boundary will be receiving marginal signals. Therefore, plant drift will cause these boundary amplifiers to fluctuate between reporting OK and reporting an error condition. In general, though, the size of the cluster will not decrease over time, with the exception of the boundary amplifiers, since faults typically become worse over time. Decreasing cluster size, though, may be seen for faults whose observability is effected by plant drift. For example, if an amplifier's nominal values are not set properly, the amplifier will report an error each time the plant drift causes the signal levels to drop below the nominal settings. As the plant drift causes the signal levels to increase, these error messages from the amplifier cease, thereby, causing the generated fault cluster to collapse only to be recreated during the next drift cycle. For most other types of faults, the associated cluster will only collapse once the fault has been repaired.

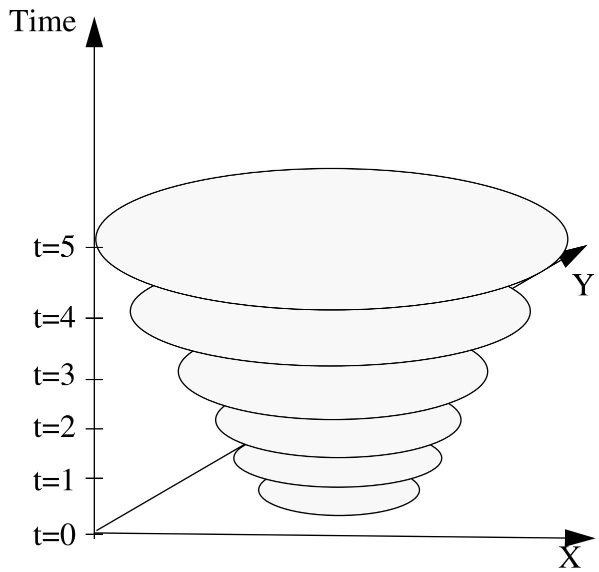


Figure 4.3: Temporal and spatial growth of a fault cluster.

At any given time, the network will contain multiple, independent fault clusters each associated with a particular network fault, as shown in Figure 4.4. These clusters are not necessarily isolated from each other, but may have common elements and, in fact, one cluster may be a subset of another. For example, if a particular section of the network is experiencing both a failed power supply and a failed amplifier then the two clusters associated with these two failures will contain common amplifiers, as shown by the overlapping of clusters A and C in Figure 4.4. This definition of a cluster can only be applied to the network if the cause of the given fault is known a priori since, if the cause of the fault is unknown, it is impossible to precisely determine which error messages can be associated with which faults. This is particularly true in the case of multiple, overlapping clusters.

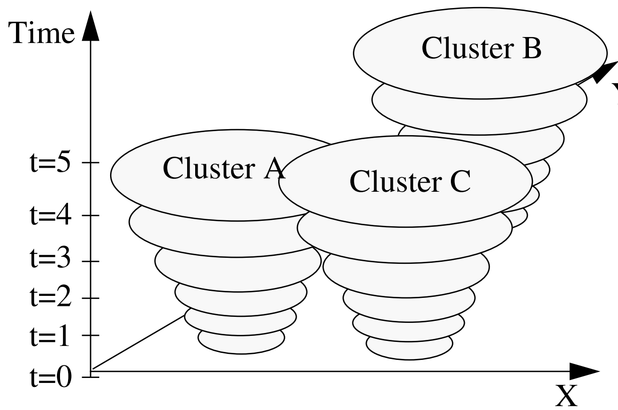


Figure 4.4: Multiple fault clusters occurring within network.

One advantage of using this fault cluster model is that it allows a means of filtering the large number of fault messages into a relatively small number of clusters. The diagnostic procedures can then be applied only to this small number of clusters, instead of being applied to each of the individual error messages in turn. Since a large number of amplifiers may be affected by a single fault and these amplifiers may continue to issue error messages until the particular fault is repaired, this use of fault clusters greatly reduces the number of diagnoses that must be performed. In addition, the use of these fault clusters allows a general cluster diagnostic tool to be developed which can diagnose faults regardless of where they actually occur in the network. The fault diagnosis tool only needs to understand general cluster topology, not the specific network topology associated with the amplifiers in the neighborhood of each fault. A fault cluster also contains all the information, in the form of the SMT information of its associated amplifiers, necessary to diagnose the given fault since, by definition, only those amplifiers included in the cluster are effected by the fault. Forming fault clusters, therefore, provides a means of identifying the pertinent information needed in the fault diagnosis process, at least for network level faults. It should be noted that not all of the information contained within the clusters may be necessary to perform the diagnoses. Redundant and irrelevant information may also be included within the clusters depending on how the given faults effect amplifiers in their neighborhood. The process of forming fault clusters, though, does ensure that the minimum amount of information needed to isolate the fault will be available to the fault diagnostic process.

4.3 Cluster Server

The cluster server, as mentioned previously, is the initial stage of the fault diagnosis procedure and implements a clustering algorithm based on a modified, recursive definition of a fault cluster. This modified definition defines a fault cluster as the interconnected set of neighboring amplifiers which all report either a FAILURE, WARNING, or ALARM error message to the SMT software within a specified time window. The boundary of the cluster is defined as the first amplifiers along the search path which report back OK's to the SMT software. The search path begins at the first amplifier to report an error condition and expands outward along all the *connected* amplifiers. The term connected refers to those amplifiers which are either located within one cable span of the current amplifier or receive power from the same power supply as the current amplifier. Figure 4.5 illustrates this concept by showing the amplifers which are defined to be connected to E100004. This definition remove the need to know a priori which amplifiers are affected by a given fault.

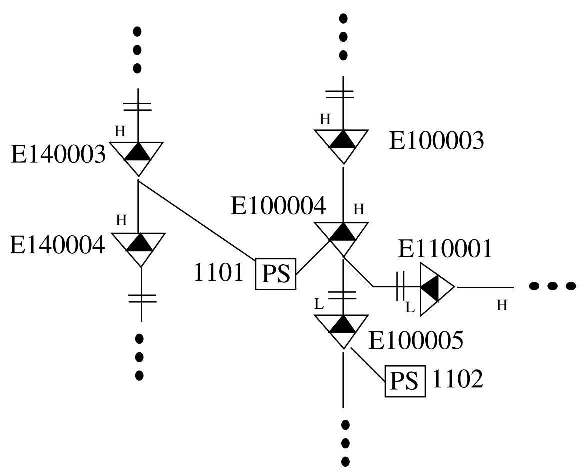


Figure 4.5: Illustration of amplifiers connected to E100004.

For the most part, this definition produces fault clusters that are identical to those that would be produced by using the original definition but without the requirement that the actual cause of the fault be known a priori. The difference in these definitions appears when multiple faults occur affecting common amplifiers. For example, in the case of the occurrence of both an amplifier and a power supply failure presented in the previous section, the modified cluster definition will cause one large cluster to be generated instead of the two clusters generated from the original definition. This large cluster will contain amplifiers which are reporting error conditions due to both network faults, and will in fact be the union of the two clusters formed by employing the original definition. Since the purpose of this work was to produce a prototype diagnostic tool, a simplifying assumption was introduced that stated that the generated cluster would be assumed to be caused only by a single network fault. The occurrence of multiple faults within the same network neighborhood appears to be a relatively rare event from a cursory look at the collected LOG and QA dump data files, therefore, this assumption should not influence the diagnostic abilities of the prototype tool. For the remainder of this work the term fault cluster will refer to this second definition which does not rely on a priori information.

The cluster server, itself, is actually a collection of software routines which provide the basic functionality necessary to implement the fault cluster definition. For this prototype diagnostic system, the server is limited to operating only on the information stored within the LOG data files stored on disk. The user specifies, in terms of command line parameters, the specific fault message for which the server is to generate a fault cluster. The server then calls upon its clustering algorithm to determine the amplifiers which are affected by this given fault. The resulting cluster is then passed onto the diagnostic system to resolve the cause of the given fault cluster.

Because the prototype diagnostic system relies on data files stored on disk instead of real-time network status information, it is extremely difficult for the user to identify the first fault message that was produced for a given fault cluster. For this reason, the user must also specify a time window, in terms of a start time and offset, within which the fault cluster is to be generated. Fault messages within the LOG data file with time stamps outside of this cluster window are automatically ignored by the clustering algorithm. It is, therefore, possible for the clustering algorithm to return a fault cluster which does not include all of the amplifiers that were affected by the given fault. If the missing amplifiers are located in the section of the network in which the fault actually occurred, then the fault diagnosis process will be unable to determine the actual cause of the fault. Currently, this problem is avoided by judiciously selecting fault messages for which to produce fault clusters. This problem will be eliminated when the system is upgraded to utilize real-time data since the first reported error message for a given cluster will be easily determined.

4.3.1 Cluster Algorithm

The actual implementation of the cluster definition is performed by the algorithm . This algorithm begins by defining the *seed* amplifier and the *observation window* as specified by the user within the cluster server. This seed amplifier is then entered as the first element of the *suspect queue*. All the amplifiers connected to the seed amplifier are then visited in order to determine if they were functioning correctly during the observation window. This determination is made by accessing the information contained in the LOG data files for the given amplifier during the given time window. Any amplifiers deemed to be operating in an error condition during this time period are entered into the suspect queue for further processing and, also, placed in the *failure set*, identifying them as being part of the failure cluster. Processing then continues recursively until the suspect queue is emptied. As in the definition, the boundary amplifiers are determined as being the amplifiers for which no connected amplifier was in an error connection, excluding those connected amplifiers already in the failure set. Since an amplifier connected to a failed amplifier may already be in the failure set, a check must be made to ensure that amplifiers are only added once to the failure set. Once this algorithm completes, all the amplifiers that are part of the given failure cluster will also be members of the failure set, for the specified time window. In order to determine the topological information necessary to determine connected amplifiers, this algorithm utilizes the topology server and its associated network data file. The information in the LOG data files is accessed by the algorithm through the use of the associated LOG file access routines.

Clustering Algorithm:

Define *seed* amplifier
Define the *observation_window* as [*seed_time* - *offset*, *seed_time* + *offset*]

Put *seed* into *suspect_queue*

While the *suspect_queue* is not empty

A: Remove first element from *suspect_queue*.;
name it *current_amplifier*
If *current_amplifier* is in the *failure_set* then discard it and goto A:
If *current_amplifier* does not register a FAIL or
WARNING or ALARM within the *observation_window*
then discard it and goto A:
else
find all amplifiers connected to the *current_amplifier* and
place them in the *suspect_queue*
place *current_amplifier* in *failure_set*

End when *suspect_queue* is empty.

4.3.2 Modified Algorithm for Low Pilot Amplifiers

The previously presented algorithm implements the cluster definition reasonably accurately except for particular types of faults involving low pilot amplifiers. As mentioned previously, these amplifiers are interspersed between every four high pilot amplifiers and are used to amplify the low frequency cable signals. If a fault occurs such that the low frequency cable signals remains unaffected, then it is possible that the low pilot amplifier will report an "OK" condition to the SMT software even though the high pilot amplifiers surrounding the low pilot amplifier are reporting fault conditions. The previous algorithm will stop generating the cluster along a given search path once the low pilot amplifier is reached since this amplifier is reporting back an OK condition. This will result in two separate clusters begin generated by the cluster server for the same network fault, one for each side of the boundary caused by the low pilot amplifier's "OK" status. For this reason an extra check must be added to the original algorithm to check one amplifier past low pilot station[1]. This check only needs to be done for upstream and downstream amplifier connections since power grid connections are independent of the actual cable signal levels.

Modified Clustering Algorithm:

Define *seed* amplifier
Define the *observation_window* as [*seed_time* - *offset*, *seed_time* + *offset*]

Put *seed* into *suspect_queue*

While the *suspect_queue* is not empty

A: Remove first element from *suspect_queue*.;
name it *current_amplifier*
If *current_amplifier* is in the *failure_set* then discard it and goto A:
If *current_amplifier* does not register a FAIL or
WARNING or ALARM within the *observation_window*
then discard it and goto A:
else
If the parent of *current_amplifier* is a Low Pilot
include parent of *current_amplifier*'s parent in set
of connected amplifiers
find all amplifiers connected to the *current_amplifier* and
place them in the *suspect_queue*
place *current_amplifier* in *failure_set*

End when *suspect_queue* is empty.

No special processing is required for the medium pilot amplifiers since their pilot frequencies are close enough to those of the high pilot amplifiers that any network fault causing the high pilot to report an error condition will also cause an error condition to be reported to by the medium pilot amplifier. Therefore, there will be no "OK" amplifier to act as a false boundary for the fault cluster as was the case for the low pilot amplifiers.

1. In rare cases, this algorithm may cause two independent clusters to be combined. This will only happen when two fault clusters occur at the same time and have the same low pilot amplifier at the cluster boundary. No such cases have currently been identified from the collected data.

4.3.3 Testing

For the purposes of testing the cluster server software, the server was modified to automatically generate a list of all the fault clusters associated with all the fault messages stored in the LOG files for a four month period starting in January 1992. The time window offset was specified as 15 minutes. Since the data transmitted to UVic currently only contains every other polling cycle, this offset represents the time required to perform two complete SMT polling cycles within this data set, one either side of the given fault message start time. The specification of the time offset in this manner ensured that at least one complete fault cluster was generated for each of the network fault which occurred during this 4 month time period. The net result of this batch run of the cluster server was that over 5000 fault clusters were identified within the given time period. Since, as mentioned previously, a given fault produces a large number of associated fault messages by the affected amplifiers, the majority of network faults within this time period are represented by multiple fault clusters. Therefore, these 5000 fault clusters represent on the order of only several hundred actual network faults. This approximation is obtained by assuming an average fault occurrence rate in the order of 10 per day which appears to be in rough agreement with the failure rates observed by Rogers' personnel. A small number of these clusters were then randomly selected and compared manually to the LOG data files in order to determine the correct operation of the server software and specifically the cluster generation algorithm. Although this method of verifying the software does not guarantee its correct operation in all cases, it does provide an intuitive indication that the clustering software does function correctly the majority of the time.

4.3.4 Limitations

Although the current version of the cluster server does appear to operate correctly, it is limited in several important ways. Primarily, the current version of the software takes a fair amount of time, on the order of 15 minutes, to generate the larger fault clusters. This large time delay is mainly the result of the server having to sequentially search the LOG data files in order to determine whether a given amplifier is reporting an error condition during the given time period. Currently, the entire sequential search must be repeated for each of the amplifiers in question and, hence, accounts for the majority of the delay in cluster formation algorithm. Once the diagnostic tool is upgraded to use real-time data, this problem should be resolved and allow the server to build clusters within a more reasonable time period. Another limitation of the server is that it currently has no means of filtering out fault messages for amplifiers that have already been found to be part of existing fault clusters. This results in a large amount of duplicated effort as seen by the 5000 cluster produced by the server for the January to May 1992 time period. This problem will be addressed as part of the next stage of research in this project.

Since the cluster server relies on the topology server, and hence the network data file for its understanding of how the amplifiers are interconnected, any errors within this file will result in incorrect cluster formation and, therefore, invalid fault diagnoses. This network data file, therefore, is one of the key elements affecting the diagnostic accuracy of the complete tool because errors within it are amplified by the cluster generation process. This issue of topological accuracy is currently being addressed within Rogers. The current version of the server software also implicitly assumes only single faults exist within the generated clusters due to how the cluster definition is implemented within the clustering algorithm. As the entire diagnostic tool evolves over the course of the project, this limitation will eventually need to be addressed, and this will probably take the form of allowing the cluster boundaries to be re-defined once partial fault diagnosis has been performed. This assumption was made since the majority of the fault clusters within the currently collected data appear to be caused by single network faults.

The final limitation of the cluster server comes from the fact that the C-COR SMT software only makes LOG file entries for a given amplifier when that amplifier's condition changes. Since the cluster server relies on these LOG file entries in its determination of an amplifier's current state, it is possible that an amplifier will not be included in the given cluster even though it is working outside of its nominal range. This problem arises because the amplifier may have dropped below its nominal levels prior to the beginning of the observation window used in the clustering algorithm and, therefore, the fact that the amplifier is undergoing a fault condition will not be observed by the algorithm and the amplifier will be assumed to be OK. This problem will need to be addressed in subsequent versions of the server by providing the server with a means of verifying the amplifier's status through the use of the QA dump data. This ability could not be included in the current version since it requires that the nominal levels for all the network amplifiers be known. This information is currently unavailable.

4.4 Cluster Diagnostic Software

Once the cluster server finishes determining which amplifiers are affected by a given network fault, the cluster and the associated amplifier status information is passed on to the cluster diagnostic software for fault identification. There were basically three methods by which this section could be implemented, namely as a separate C program, as a rule base within a commercial expert system shell, or as a rule base within a custom expert system shell. It was decided that utilizing a commercial expert system shell would be the most productive of these methods to employ at this early stage in the diagnostic tool development, since this commercial system provides rich developmental and extensive debugging tools. The commercial shell that was chosen was NEXPERT OBJECT and this was selected based on its features, its relatively low cost, and the ease with which it can be integrated within existing C programs. Using either a custom expert shell or a C program to perform the cluster diagnosis was not a viable option since they both entailed long development times and would have lacked the high level debugging tools useful for verifying the diagnostic operations.

Even though NEXPERT is capable of making use of external C routines within its inference engine, this interface was found not to be robust enough to allow NEXPERT to access the information contained in the cluster and topology servers. For this reason, this information had to be explicitly imported into NEXPERT's knowledge base. In addition a rule base had to be created within NEXPERT to instruct the inference engine about how to perform cluster fault diagnosis. The remainder of this section will describe these two parts of the tool's diagnostic section and provide an example cluster diagnosis.

4.4.1 Cluster Representation

NEXPERT utilizes an object oriented paradigm to represent factual knowledge within its knowledge base. This paradigm allows for hierarchical data structures to be represented in terms of class, object, and slot constructs. Once the cluster server has established the boundaries of a given fault cluster, it then issues the appropriate NEXPERT calls to load the topological and status information relating to the given cluster as instantiations of pre-defined network classes. These classes were termed: the topology data class, the log data class, and the amplifier data class. Figures 4.6, 4.7, and 4.8 illustrate the instantiation of objects from these three classes for a particular amplifier within a given fault cluster. The topographical data object contains the topographical information about the amplifiers obtained from the topology server.

As can be seen from the figures, the topology data object contains the amplifier's topographical information as stored in the topology server, and the log data and amp data objects contain the LOG file data and QA dump data, respectively, for the time period for which the cluster was generated. Each amplifier within the cluster is represented in terms of instantiations of these three classes. In addition a fourth class, termed the amplifier class, was created as a superset of these three lower level classes to allow all the information about a given amplifier to be accessed through a single point. Figure 4.9 shows the complete instantiation of a particular fault cluster within NEXPERT's knowledge base. These instantiated objects completely quantify the given fault cluster in terms of its structure and behaviour within the observation window for which it was generated.

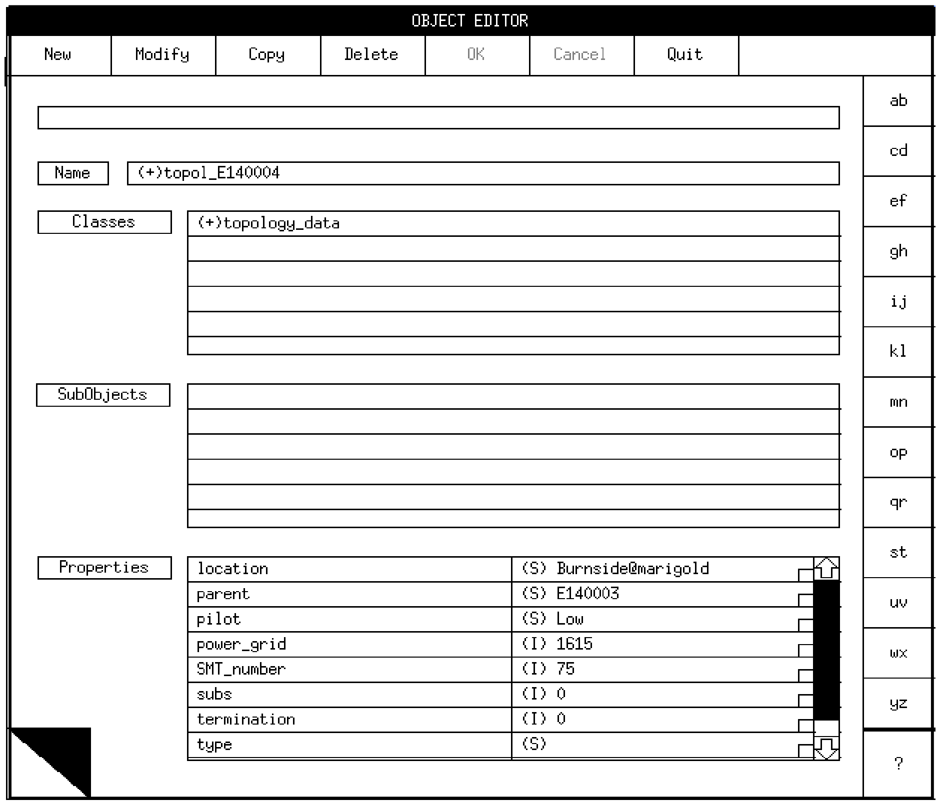


Figure 4.6: Instantiated topology data object.

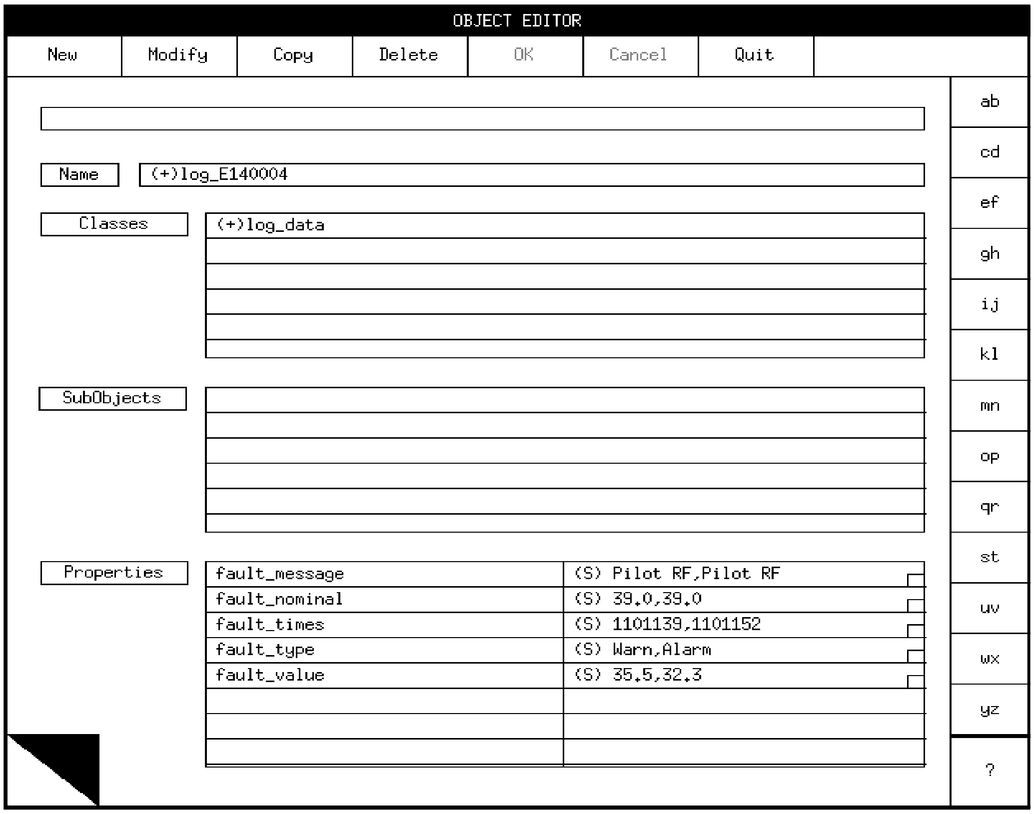


Figure 4.7: Instantiated log data object.

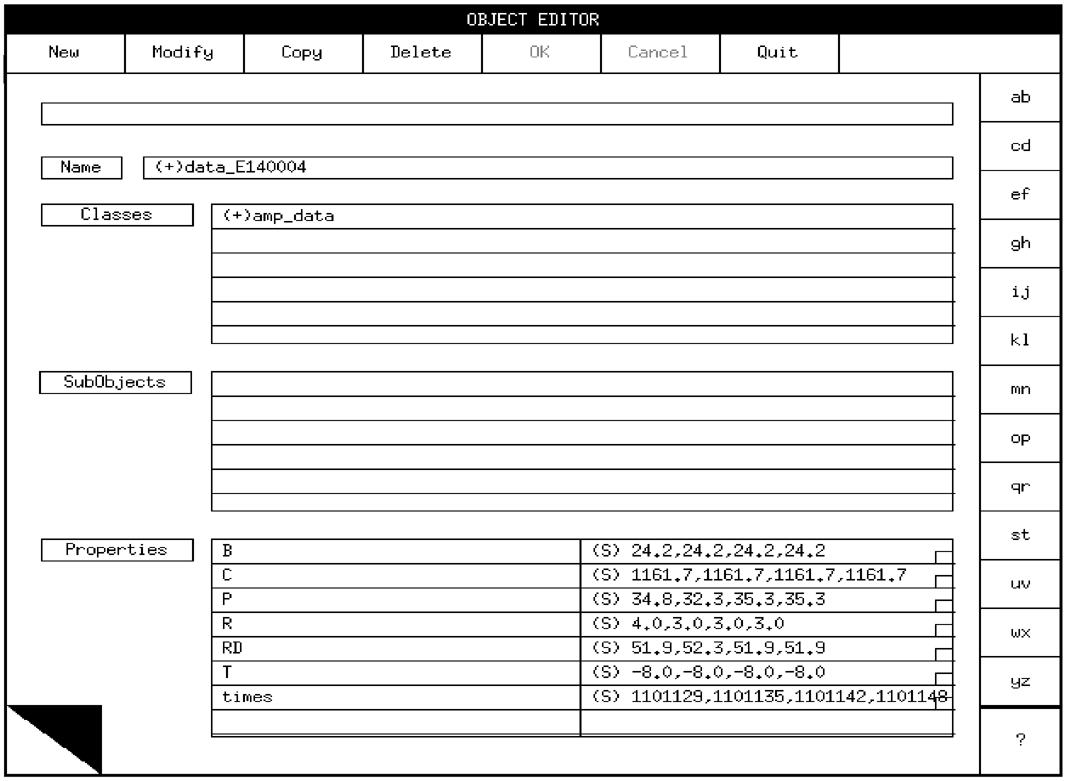


Figure 4.8: Instantiated amp data object.

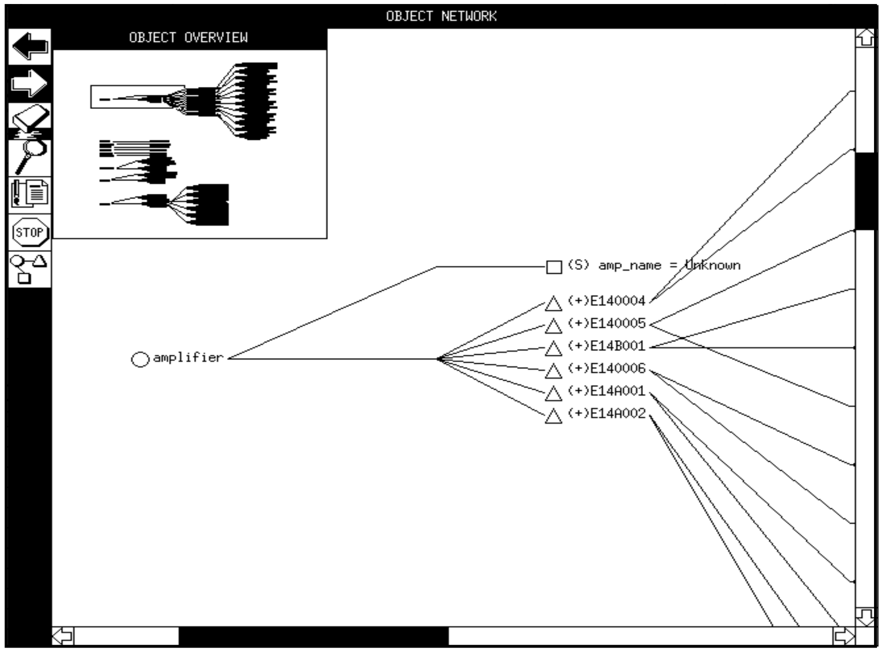


Figure 4.9: Complete cluster instantiation of example fault cluster.

4.4.2 Knowledge Processing

The knowledge processing section of the cluster diagnostic software can be viewed abstractly as a decision tree constructed from the domain expert's network fault isolation procedures [23]. This decision tree utilizes the structural and behavioural knowledge stored in the fault cluster representation to systematically isolate the possible cause or causes of the observed fault. This process can be viewed as starting with a set of all possible fault causes and slowly eliminating causes at each node within the tree as they are traversed. The set which remains once a leaf node is reached represents all the possible faults that could have caused the observed features in the fault cluster. Leaf nodes may contain more than one fault cause, indicating that there are several possible causes of the given fault and that the diagnostic system is incapable of further distinguishing between them. Figure 4.10 illustrates the structure of this network level diagnostic tree.

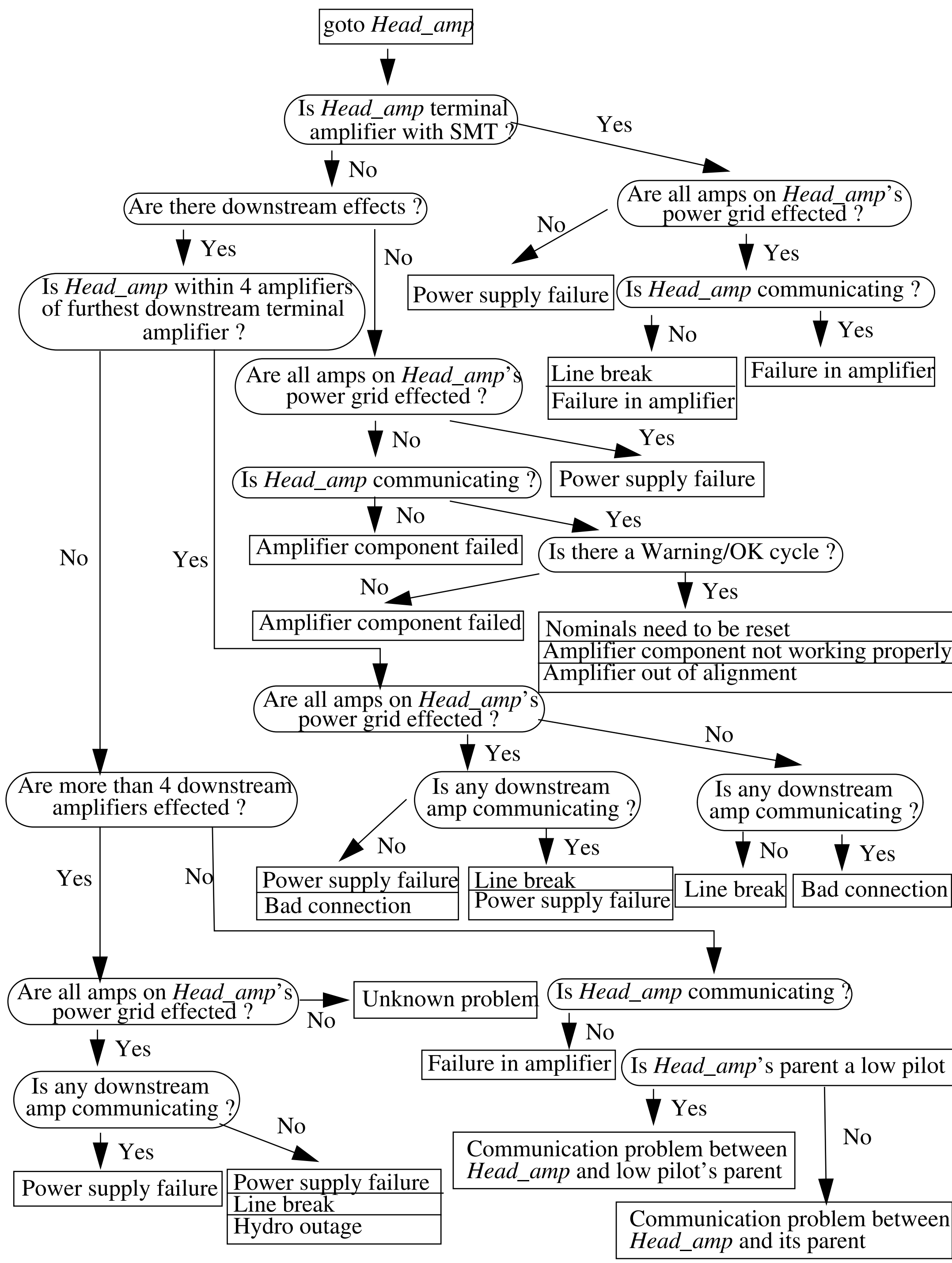


Figure 4.10: Network level diagnostic tree.

Several important aspects of the diagnostic process can be seen from this figure. Primarily, the first step in the process is to locate and identify the head amplifier of the fault cluster. This head amplifier is defined as the amplifier furthest upstream which reported an error condition to the SMT software. Within this context, the upstream amplifiers do not include those amplifiers which are connected over the power supply grids and, therefore, this definition will only identify a single amplifier as being the head amplifier. The parent of this amplifier will be the last amplifier in the given branch to still be reporting an OK status. Because of the nature of cable trunk amplifier networks, this head amplifier will be the amplifier where the given fault is occurring, under the one failure per cluster assumption. The diagnosed cause of the fault, therefore, refers to head amplifier or its associated upstream cable span or power supply. The figure also illustrates how the diagnostic process is based on the number of amplifiers that are affected by the given fault and, hence, included in the fault cluster. The set of all possible fault causes can be fairly effectively divided along the lines of the maximum number of downstream amplifiers affected by a given fault due to the network's structure and the varying fault severities. If the network faults are ordered according to their severity, in terms of the number of affected subscribers, then this list will also be ordered according to number of affected downstream amplifiers. Therefore, by focusing on the number of affected downstream amplifiers, this procedure inherently provides a means of identifying the severity of the given fault. The figure also illustrates the overhead that is involved in the diagnostic process to deal with boundary effects, both in terms of processing concerning head amplifiers near the cluster boundaries and head amplifiers near the boundary of the trunk amplifier network itself.

Table 4.1: Diagnosable Cluster Faults:

Cluster Fault	Description
Amplifier component failed	A component within the head amplifier has failed, but the whole amplifier has not failed since not enough downstream amplifiers are effected.
Amplifier component not working properly	A component within the head amplifier is failing intermittently.
Amplifier out of alignment	Warning/OK cycle caused by amplifier being out of alignment.
Bad connection	A poor connection between the head amplifier and its parent causing attenuation of high frequency signals.

Table 4.1: Diagnosable Cluster Faults:

Cluster Fault	Description
Communication problem between head amplifier and low pilot's parent	Fault lies in the two cable spans between head amplifier and its parent's parent. (Only applicable when the head amplifier's parent is a low pilot)
Communication problem between head amplifier and parent	Fault lies in cable span between head amplifier and its parent.
Failure in amplifier	Failure within head amplifier has caused amplifier to fail completely.
Hydro outage	Power failure is due to Hydro outage.
Line break	Line is broken between head amplifier and its parent.
Nominals need to be reset	Head amplifier's nominal values are invalid or set too narrow to allow for network drift.
Power supply failure	Fault is caused by failure in head amplifier's external power supply.
Unknown problem	Cause of fault could not be determined.

Table 4.1 lists all of the fault causes that the current version of the software is capable of diagnosing and their associated meanings. As mentioned previously, the diagnostic tree does not, in general, provide an exclusive distinction between all these possible fault causes. Instead, it indicates all of the causes which could have possibly caused the observable features in the given fault cluster. Domain experts, one the other hand, are able to differentiate between the plausible causes through the use of their experiential knowledge and by utilizing additional information sources not currently uncooperative into the diagnostic tool. Currently, once the diagnostic processing distinguishes these possible fault causes, no further diagnostic processing is performed and the possible causes are reported to the user. In future versions of the software, it may be possible to use heuristics to provide this further level of distinguishing information. Currently all of the fault diagnoses are performed through the implementation of this one fault diagnosis tree and, therefore, only a small set of common network level fault causes can be identified. This set, though, appears, through the experimentation that has been

performed, to cover the majority of the faults which occurred in the January to May 1992 time period for which the diagnostic tool has been tested. This tree also currently only uses information obtainable through the topology server and the LOG data files.

This diagnosis tree was implemented as a NEXPERT rule base and associated external C routines. The remainder of this section will present a description of these two parts of the diagnostic tree implementation.

4.4.2.1 Diagnostic Rule Base

The diagnostic rule base is implemented utilizing NEXPERT's if-then rule constructs and implements the majority of the functionality required by the network level diagnostic tree. This rule base is composed of approximately 55 separate rules and is processed through forward chaining by NEXPERT's inference engine to arrive at the cause of a given fault cluster. Because the diagnostic processing is performed within NEXPERT a certain portion of this rule base is involved in maintaining housekeeping variables within the knowledge base. As much as possible, inherent constructs within NEXPERT, such as multi-valued fields, were utilized to minimize the complexity of this rule base. Figure 4.11 shows this diagnostic rule base as it appears within NEXPERT's rule base editor.

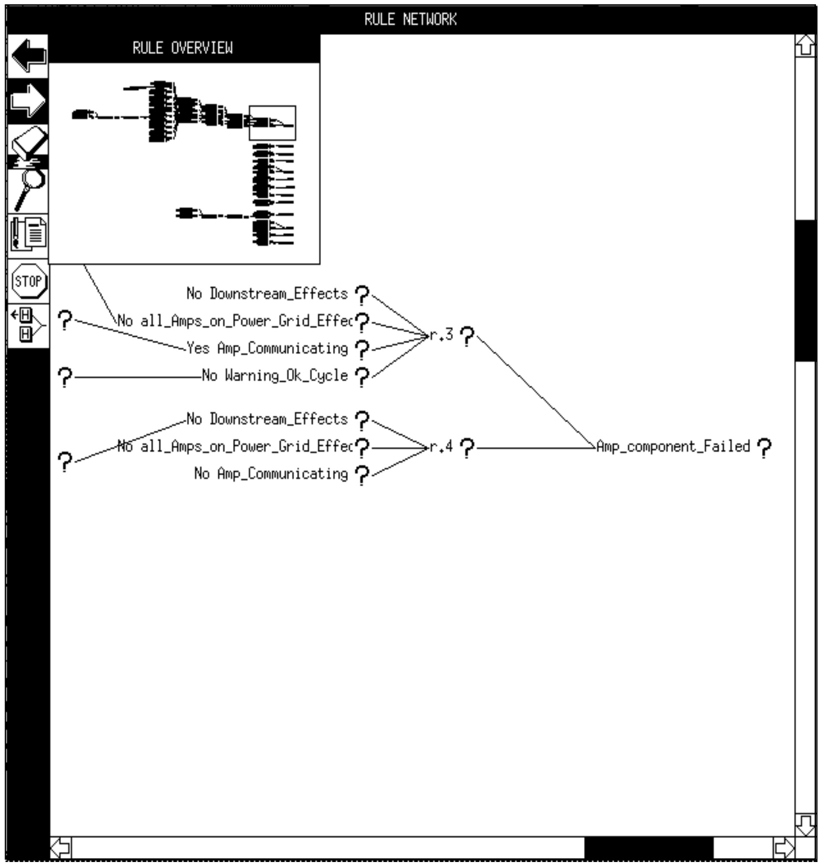


Figure 4.11: Diagnostic rule base.

4.4.2.2 External C Routines[2]

NEXPERT's internal constructs for operating on the loaded cluster representation were not powerful enough to adequately answer some of the questions posed within the diagnostic tree. For this reason a set of external C routines had to be created to provide these answers automatically. These routines mainly involved using a looping construct to

2. The majority of these routines were written by Mr. Andrew Watkins and they are included in this work for completeness.

traverse amplifiers within the cluster structure. This type of construct is outside of the expert system paradigm and, therefore, unavailable within NEXPERT's domain. Table 4.2 illustrates the main C routines that were developed and there associated functions.

Table 4.2: Table of External C Routines.

Routine	Description
CSmeasure_downstream_effects	Counts the number of amplifiers downstream of the head amplifier that are part of the given fault cluster.
CSget_head_amp	Finds the head amplifier in the fault cluster.
all_amps_on_PG_loaded	Determines if all the amplifiers connected to the specified power supply are part of the given fault cluster.
TSget_diatances_to_network_term	Finds the maximum number of downstream amplifiers between the head amplifier and the network boundary.
MMVmatch_fault_message	Determines if a given fault message is contained in the cluster. Currently, used only to determine if given amplifiers are communicating with the SMT software.

4.4.3 Example Cluster Diagnosis

This example cluster fault diagnosis is based on the fault illustrated in Figure 4.12 Once the cluster has been loaded into NEXPERT's knowledge base by the cluster generation software, the first step performed by the diagnostic software is to locate the cluster's head amplifier. This task is performed using one of the external C routines, namely CSget_head_amp, since it involves traversing the amplifiers within the cluster and the traversal itself starts at the amplifier who's error report caused the cluster to be generated. In the case of this particular fault the first reporting amplifier is E10000G and this traversal involves just visiting E10000G's parent, E10000F, which is the amplifier furthest upstream in the cluster and, therefore, is the cluster's head amplifier. Once the head amplifier is found, NEXPERT's inference engine then determines whether the cluster has been generated at the boundary of the cable network, defined as being the last amplifier in a branch having an SMT module. Since in this case the fault is not occurring at a boundary the diagnosis proceeds by determining whether the given fault is generating any downstream effects or whether the effects of the fault are only felt by the head amplifier.

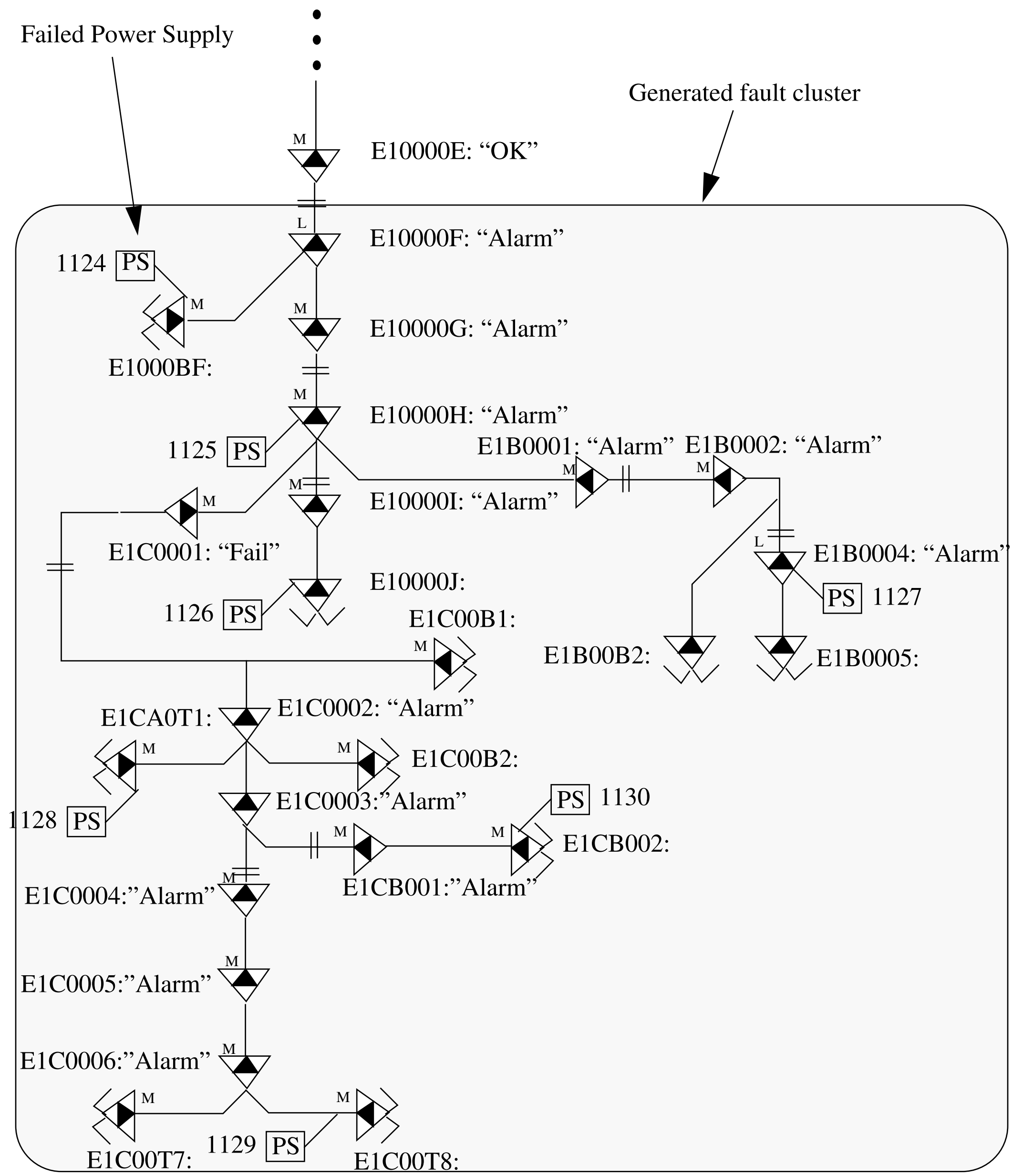


Figure 4.12: Example network fault

In this case, downstream amplifiers are affected so the diagnosis proceeds by determining whether the head amplifier is within 4 amplifiers of the cluster's boundary. Since it is not, the diagnostic system is then able to determine whether more than 4 downstream amplifiers are affected by the fault. The answer to this question can be used to indicate the severity of the network fault since it differentiates between the number of affected subscribers. In this particular cluster, more than 4 downstream amplifiers are effected and the diagnosis process has now eliminated all the possible fault causes except for a power supply failure, a Hydro outage, a line break, and an unknown problem. The diagnostic system then proceeds to further eliminate some of these possible fault candidates by determining whether all the amplifiers on the head amplifier's power grid are also effected by the fault. In this case, E10000F's power supply also powers E10000G and E1000BF, of which only E10000G is included in the fault cluster since E1000BF does not have a status monitoring transponder. Since both E10000F and E10000G are in the cluster, the answer to this question is yes and this answer indicates that there is a power supply problem occurring.

This problem may be due to either a failure in the head amplifier's power supply, or a failure in the Hydro power supplying this power supply, or a line break in the cable span over which this power is sent from the power supply to the head amplifier. The final question attempts to isolate whether the failure is actually due to the power supply by determining whether any downstream amplifiers are still communicating with the SMT software. If they are, then this eliminates line breaks and Hydro outages as being possible fault causes. A Hydro outage is eliminated since such failures typically effect several power supplies and, therefore, cause the cable signal to drop below the level required for SMT communication. In this case, all downstream amplifiers are communicating and, therefore, the fault is determined to have been caused by a power supply failure in the head amplifier's power supply. Figure 4.17 illustrates the path this decision took along the network diagnostic tree and indicates answers obtained through calls to the external C routines.

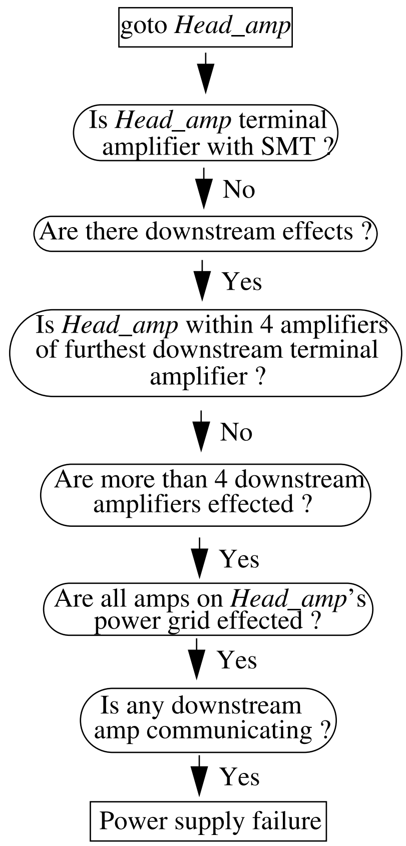


Figure 4.13: Path of diagnosis through the network level diagnostic tree.

4.4.4 Testing

The diagnostic software was tested by diagnosing a set of selected fault clusters, and comparing the diagnosis generated by the system with that produced by the domain expert given the same behavioural and structural information. The results of these tests are presented in the following chapter. The rule base and associated C routines were also tested by explicitly executing each rule to ensure that these two parts accurately implemented the diagnostic procedure as outlined by the diagnostic tree. Any deviations from the desired diagnostic procedure were isolated and corrected and the entire rule base was then reexecuted until no deviation from the expected diagnostic path were detected. The diagnostic tree was also reviewed by Rogers' domain expert to ensure that it accurately represented the methods employed by the expert in network level fault isolation.

4.4.5 Limitations

As mentioned previously, the current diagnostic system is limited to only diagnosing single faults within any given fault cluster. Some of the generated clusters, though, will be the result of multiple faults occurring simultaneously in the same neighborhood of the network. The occurrence of multiple, neighboring failures appears to be fairly rare as indicated by the LOG and QA data files for the January to May time period. For these types of clusters, the diagnostic system will only diagnose the fault occurring furthest upstream since the downstream fault(s) will be masked by this first fault. These downstream faults will not be diagnosable by the current system until the original fault has been repaired. The current system may also mis-diagnose faults for multiple fault clusters since the downstream faults will cause the system to mis-interpret the number of amplifiers affected by the fault furthest upstream. These problems will need to be addressed in future versions of the diagnostic software.

The current system also diagnoses only a relatively small number of network level faults and these faults may not represent the complete set of possible network faults. Further research into the nature of faults within the cable network is needed to determine the fault coverage provided by the current system. This is part of the normal development cycle of expert systems in which a prototype is developed and tested by domain experts and then new rules are incrementally added as the limits of the systems's knowledge domain are reached. Faults occurring outside of the network level are not diagnosable by the current system, but these can be easily introduced to the system by adding different diagnostic trees containing the desired diagnostic procedures and information. Only topological and LOG data information are used in the current system, and, although this information does provide sufficient differentiation between the possible fault causes for the prototype system, more detailed diagnoses may be attainable through the use of the QA data. Since the current system was designed as a prototype diagnostic system, these limitations are relatively minor and relate mainly to how the system will have to be improved in order to extend its diagnostic abilities.

4.5 Chapter Summary

This chapter presented a detailed overview of the prototype diagnostic system that was created. This overview included a discussion of the theory behind the use of fault clusters and how these clusters dynamically change within the cable trunk amplifier network, and illustrated how the complete system was implemented. In addition, examples of both the cluster generation and cluster diagnosis processes were presented along with detailed discussions regarding the testing and limitations of these two diagnostic tool sections. Within the diagnostic software section, the use of the diagnostic tree was also presented along with a discussion of how this tree structure was implemented within NEXPERT OBJECT's expert system constructs.

Chapter 5:

Results

5.1 Introduction

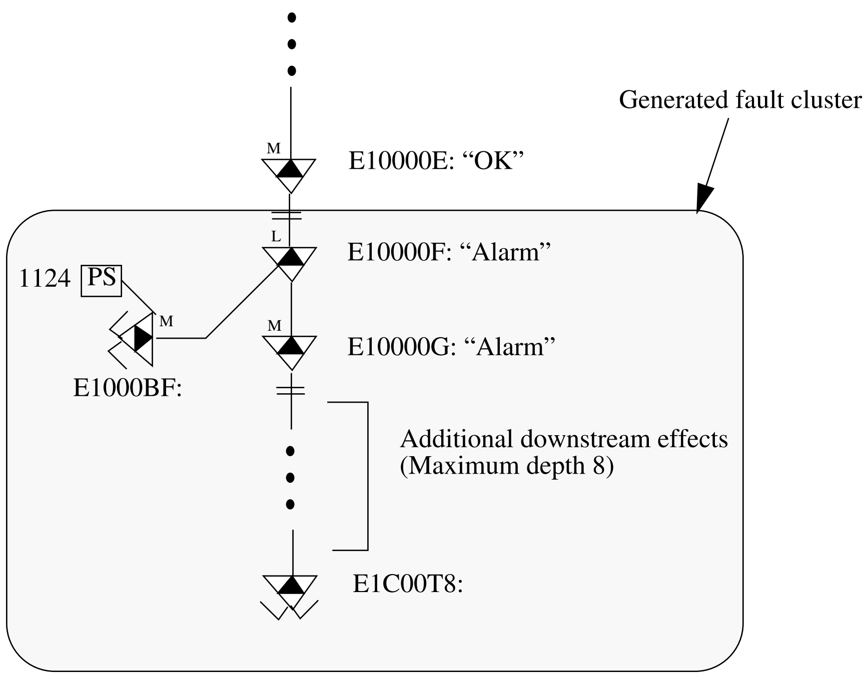
This chapter will present the results obtained from using the prototype network diagnostic tool developed during the course of this work to diagnose a set of ten network faults which occurred in the Victoria cable trunk amplifier plant between January and May 1992. In each case, the diagnosis obtained by the tool will be compared to the diagnosis obtained from the domain expert and any differences will be discussed. These results will be reported according to the domain expert's diagnosis. These ten cases were chosen from the approximately 5000 fault clusters generated for the given time period in such a manner as to present the diagnostic tool with interesting faults, in terms of a relatively large number of affected amplifiers, and with faults within its diagnostic expertise. Single amplifier clusters were not presented to the diagnostic tool since these types of faults generally are caused by internal faults within the amplifier and, hence, are not within the network level domain of the tool. This set of ten faults is not meant to provide a complete test of the diagnostic tool, but instead to provide a proof of concept for the use of expert system based diagnostic tools in the diagnosing network level faults within trunk cable amplifier networks.

All of the test samples had to be applied to the diagnostic tool manually by specifying the desired fault in terms of the first amplifier affected and the time of occurrence. This data was obtained from the data file produced by running the cluster server over the LOG data files representing this four month period. Care was then taken to select specific clusters for diagnosis which included the complete set of affected amplifiers. This was necessary since, as was mentioned previously, a 15 minute offset was used as the specified time window and, therefore, if the fault existed for longer than 15 minutes it was possible for there to be clusters for which the entire set of affected amplifiers were not included. These clusters could not be used to test the diagnostic tool since they did not contain the complete set of information about the network behaviour caused by the fault and, hence, could not be accurately diagnosed. This problem is an artifact of the current stage of the implementation of the diagnostic tool. In future versions of the tool, the faults generating these clusters will be automatically filtered out since they represent fault messages describing the behaviour of a fault which has already been diagnosed.

In order to allow the domain expert to also diagnose the given set of faults, it was necessary to manually collect all of the pertinent information from the various stored data files and to produce a graphical representation of the fault cluster. The domain expert was then required to come to the lab at the University of Victoria to perform his diagnosis. As can be seen, this resulted in a relatively time intensive verification process and, hence, limited the number of faults which could be processed in this manner. In addition, in some cases the available information concerning a given fault was not sufficient for the domain expert to provide a diagnosis and, hence, no comparison could be made to the diagnosis obtained from the diagnostic tool. These clusters are, therefore, not included within this chapter and are not part of the aforementioned set of fault clusters. Since Rogers does not currently keep a record of network faults and their associated repairs, the only way available to validate the rule base was to compare its results with those of the domain expert over the given fault set.

5.2 Power Supply Failures

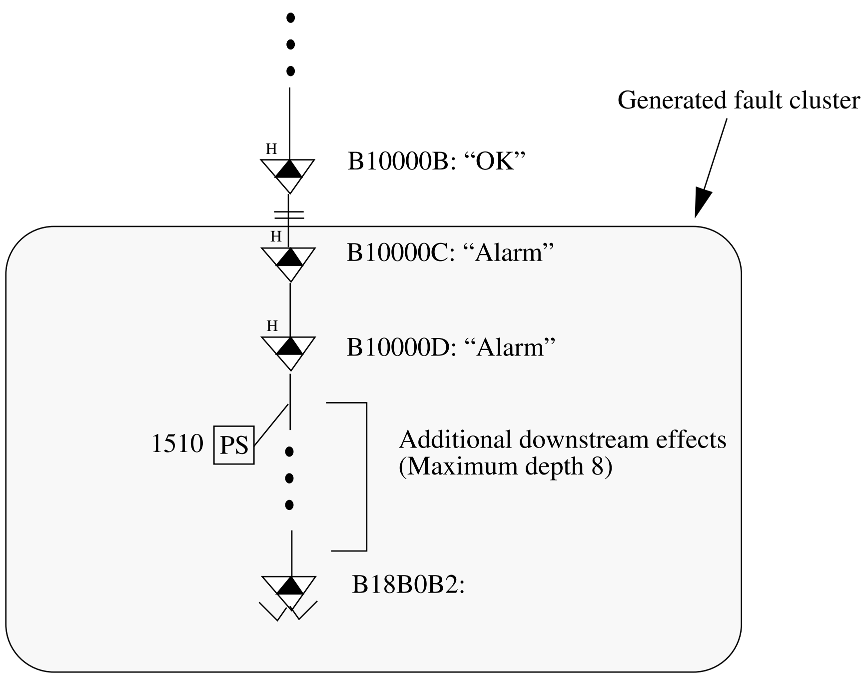
Of the set of ten fault clusters, three were diagnosed as being the result of a power supply failure in the head amplifier's external power supply by the domain expert. Figures 5.1, 5.2, and 5.3 illustrate the fault cluster generated by the cluster server and the behaviour of the network elements in the neighborhood of these fault occurrences. The existence of a power supply failure is indicated in all three cases by the large number of affected downstream amplifiers and by the sudden attenuation and slow increase of the pilot signal level from the point of the failure to the edge of the fault cluster. This steady increase is a result of the automatic level control circuitry being triggered in each of the downstream amplifiers still receiving power and allowing the amplifiers to increase the pilot level above the loss incurred by the preceding cable span. This domain expert was also able to confirm his diagnoses by identifying the power reset messages associated with the head amplifier further along the LOG data file from the fault occurrence. These reset messages are indicative of the procedure employed by the repair crews to restart failed power supplies.



Expert Diagnosis: Power supply failure in power supply 1124.

Tool Diagnosis: Power supply failure in power supply 1124.

Figure 5.1: Diagnosed power supply failure fault cluster no. 1.



Expert Diagnosis: Power supply failure in power supply 1510.

Tool Diagnosis: Power supply failure in power supply 1510.

Figure 5.2: Diagnosed power supply failure fault cluster no. 2.

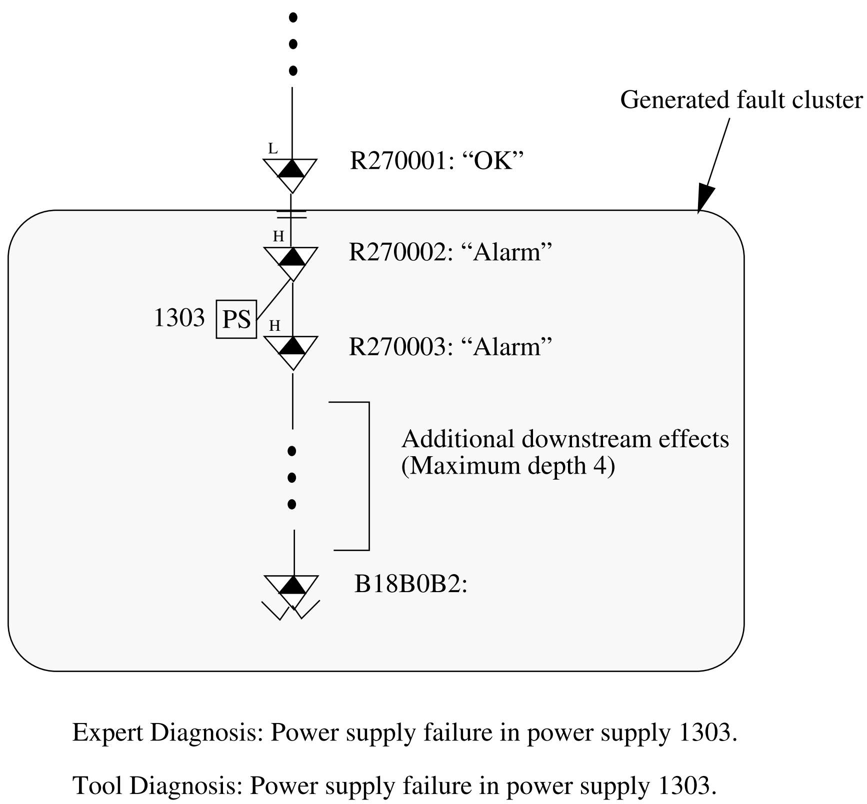
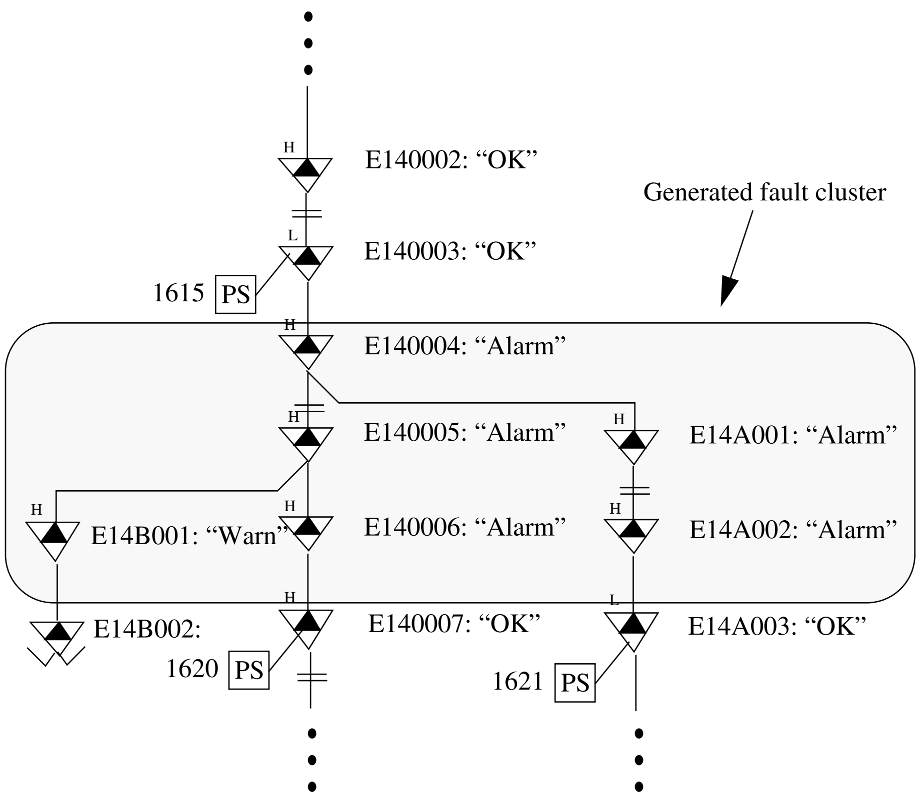


Figure 5.3: Diagnosed power supply failure fault cluster no. 3.

In each of these cases, the diagnostic tool was able to locate the head amplifier of the fault cluster and accurately diagnose the cause of the fault as being a failure in the head amplifier's external power supply. The diagnostic tool was able to accurately perform these fault diagnoses utilizing only information about the given fault clusters' structure since the current rule base does not contain rules regarding behavioural reasoning. The domain expert, on the other hand, used the structural information for the original diagnosis and then confirmed the diagnosis through the use of the behavioural information. These faults, therefore, indicate that the current prototype diagnostic tool does not completely follow the diagnostic procedures employed by the domain expert. The use of the behaviour information to verify structural diagnoses will probably be useful in later stages of the diagnostic tool's development to improve diagnostic accuracy and to more accurately follow the domain expert's diagnostic procedures. Nevertheless, the diagnostic rules employed for this class of faults were able to accurately diagnose the cause of the fault cluster in all three cases.

5.3 Bad Connection Failure

One of the set of failures was determined to be caused by a bad connection between the head amplifier, E140004, and its parent's parent, E140002. The generated cluster is illustrated in Figure 5.4 along with the associated behavioural information. This particular cause is indicated by the relatively few number of affected downstream amplifiers and by the fact that the head amplifier's parent, a low pilot amplifier, is unaffected by the fault. Together these two features do not provide conclusive evidence that the failure is due to an upstream cable connection. For example, a partial failure within the output of the port 1 amplifier module of either amplifier E140003 or E140004 could also have caused the observed network behaviour. The domain expert was able to eliminate the other possible causes of the given fault cluster by utilizing intuition obtained through past experience.



Expert Diagnosis: Water in connector between E140002 and E140004.

Tool Diagnosis: Communication problem between E140002 and E140004.

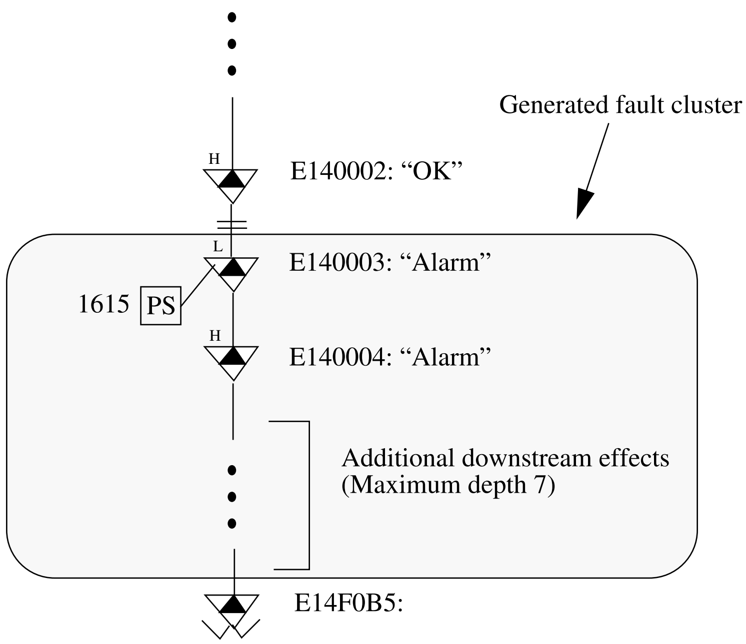
Figure 5.4: Diagnosed bad connection failure fault cluster.

The cable spans used between amplifiers are rarely composed of sections of continuous cable. Instead, each span typically contains a number of cable splices which, over time, are susceptible to water damage. Water enters the splices and causes the copper core of the coax cable to be corroded. This corrosion acts as a low pass filter to the cable signal, causing the high frequency signal to be heavily attenuated while the low frequency signals remain unaffected. The resulting network behaviour is a fault which causes the error conditions to be reported by the neighboring high pilot amplifiers and appears to leave the neighboring low pilot stations unaffected. This type of failure behaviour can also be caused by imperfections within the coax cable. For example, a creased section of cable will also exhibit the same low pass filter characteristics. Water damage occurs quite commonly in the Victoria cable plant and, hence, explains the domain expert's intuitive diagnosis. Without actual reports from the repair crew who serviced this fault, it was impossible to verify this diagnosis.

The diagnostic tool also diagnosed the fault as being caused by a bad connection between the head amplifier and its parent's parent and identified the head amplifier as E140004. The prototype tool, though, was not able to provide the more specific diagnosis of water corrosion in a cable span splice since this level of diagnostic ability is currently not supported. The fault, though, indicates that further versions of the tool may be able to present the user with fairly detailed fault diagnoses, although these detailed diagnoses will probably have to be implemented in terms of heuristics. For example in this case, a future version of the diagnostic tool may associate belief factors with the possibilities of the bad connection being caused by either a damage connector or a partial failure within the port 1 amplifier module. The tool, though, does implicitly correctly implement the domain expert’s empirical knowledge that the possibility of a port failure is relatively rare compared to the possibility of water damage.

5.4 Line Break Failure

Figure 5.5 illustrates the structure and behaviour of a failure resulting from a break in the cable span occurring between the head amplifier, E140003, and its parent, E140002. The domain expert was able to diagnose this fault from the large number of downstream amplifiers which the SMT software was unable to establish communications. Such a large communications failure indicates that the physical cable within the trunk network has been broken, there has been a wide spread Hydro outage, or a power supply failure has occurred in a particularly susceptible section of the network. Because the cable spans are not identical lengths through out the network, due to the availability of physical sites at which to locate the amplifiers, those areas containing longer than average cable span lengths are affected more severely by power supply failures. This susceptibility is due to the higher signal losses incurred over these given spans. In some areas, a given power supply may also supply more than two amplifiers on the same branch, causing failures of these given power supplies to produce significantly larger behavioural affects than would be typically expected. For this reason, Rogers is currently implementing a procedure that specifies that no more than two amplifiers on the same branch can be powered by the same power supply.



Expert Diagnosis: Line break between E140002 and E140003.

Tool Diagnosis: Communication problem between E140002 and E140002.
Hydro outage affecting power supply 1615.
Power supply failure in power supply 1615.

Figure 5.5: Diagnosed line break failure fault cluster.

The domain expert was able to isolate the cause of this particular fault as a line break, and not one of the other possibilities, by identifying the particular section of the cable network as not being one containing particularly long cable spans, or multiple in-line amplifiers with the same power supply. In addition, by the date associated with the fault messages the domain expert was able to determine that no large scale power outage had occurred in the affected area during the time period that the fault was reported. Unlike the domain expert, the prototype diagnostic tool was unable to utilize these additional information sources and, hence, provided a diagnosis that in addition to the domain expert's diagnosis that the cause of the problem was a line break between the head amplifier, E140003, and its parent, also diagnosed a power supply failure in the head amplifier's external power supply, or a Hydro outage as being possible causes of the observed network behaviour. This diagnosis illustrates the need for future versions of the tool to be able access additional information, such as cable span length, in order for the tool to use heuristics to present relative probabilities for the plausible fault causes.

5.5 Man Made Failures

Three of the set of test failure clusters, shown in Figures 5.6, 5.7, and 5.8, were caused by man made events as indicated by the presence of unit lid messages within the given LOG files associated with these faults. These messages are produced whenever repair crews open the lid of a given amplifier in order to service it. The domain expert used the occurrence of this field to diagnose each of these three clusters as actually not being caused by a fault within the network. Since the prototype diagnostic tool does not currently search for this message within the LOG files, it is incapable of filtering out fault messages caused directly by the repair crews or identifying the generated cluster as having a man made cause. Instead, the diagnostic tool proceeds to diagnose these fault clusters as if they were caused by network level failures.

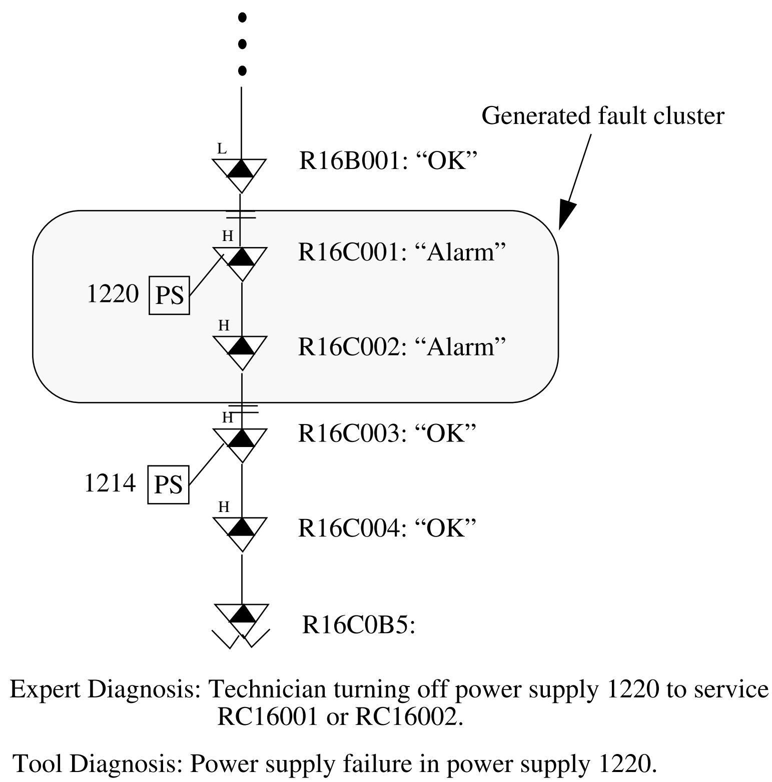


Figure 5.6: Diagnosed man made failure fault cluster no. 1.

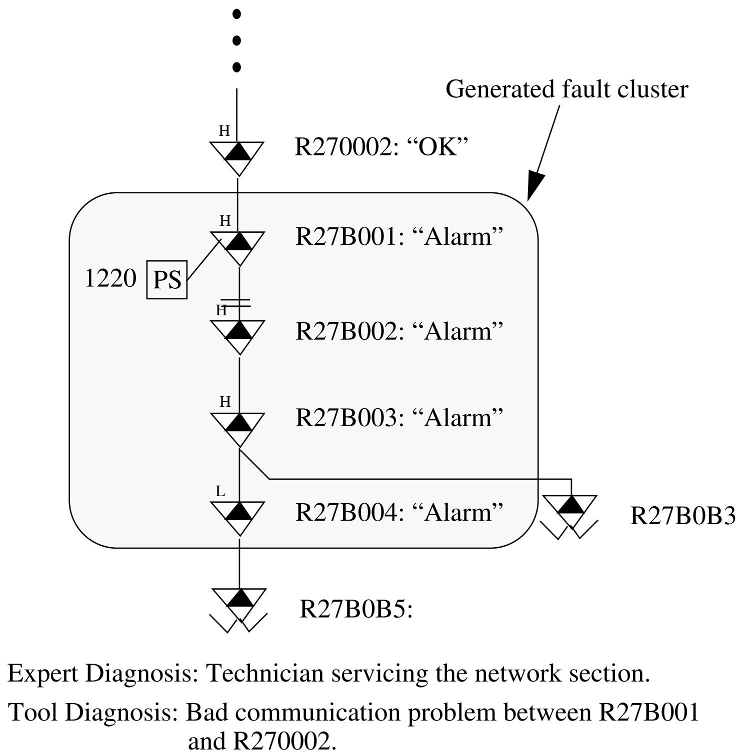


Figure 5.7: Diagnosed man made failure fault cluster no. 2.

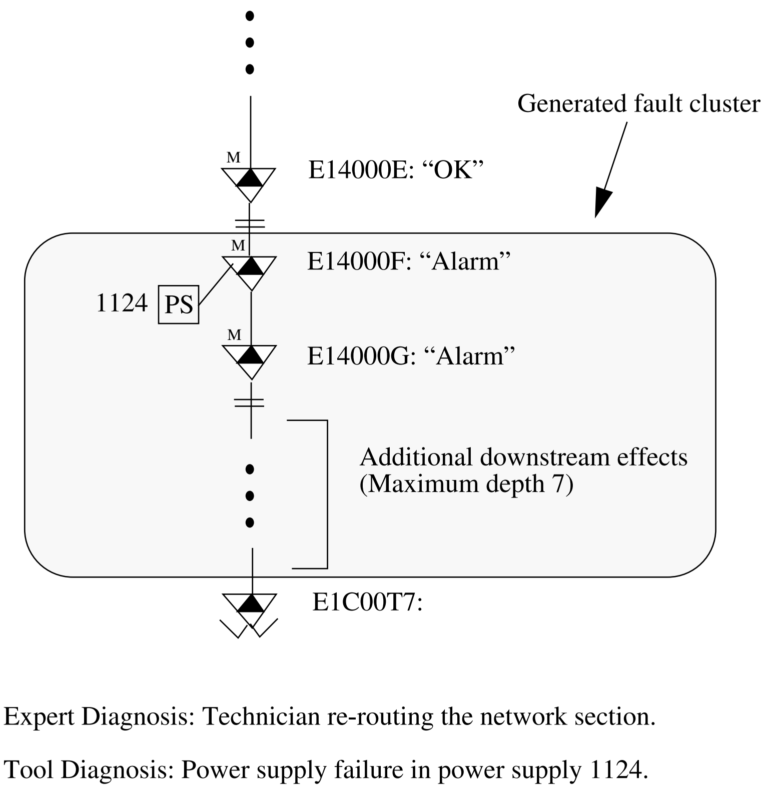


Figure 5.8: Diagnosed man made failure fault cluster no. 3

The first of these man made fault clusters, shown in Figure 5.6, was diagnosed as being caused by a bad connection between the head amplifier, R27B001, and its parent, R270002. This diagnosis resulted from there being relatively few affected amplifiers due to the location of the fault cluster at the network boundary. This fault cluster provides an indication of the special processing that is required in order to accurately diagnose the cause of faults occurring near the network boundary. The second fault cluster, shown in Figure 5.7, was diagnosed as being caused by a failure in the power supply 1124, associated with the head amplifier, E10000F. In this case, there is a power reset message in the LOG file associated with amplifier E10000F in the given fault time window. The message verifies that, during the servicing of this section of the network, the power to amplifier E10000F was interrupted during the course of the network maintenance. Therefore, the diagnosis of a power supply failure for amplifier E10000F can be considered to be an accurate diagnosis given the limited information available to the prototype diagnostic tool. The third fault cluster, shown in Figure 5.8, was also diagnosed as being caused by a power supply failure and had an associated power reset message within the given LOG file. It is interesting to note that in this case the duration of the fault was shorter than the duration of the SMT polling cycle. In general, faults which have such short durations are not diagnosable by the domain expert due to a lack of behavioural information and are, therefore, typically ignored and classed as non-events unless they occur repeatedly. The fact that the diagnostic tool was able to produce an accurate diagnosis in this case may suggest that future versions of the tool may be able to diagnose some types of faults that are currently classed as non-events. The other possibility is that this particular diagnosis was merely an artifact of the nature of the given fault and is not, therefore, indicative of a general diagnostic ability.

5.6 Unidentified Failures

The cause of the final two fault clusters presented to the domain expert could not be identified by him due to a lack of information about the behaviour of the network in the neighborhood of the fault. In the first failure, shown in Figure 5.9, this lack of information was caused by the fault being located near the boundary of the amplifier network. The domain expert was only able to suggest that the fault was probably due to a bad connection between the head amplifier, E10000G, and its parent, E10000F. The diagnostic tool was also able to produce the diagnosis of a bad connection between E10000G and E10000F. In the second failure, shown in Figure 5.10, the lack of behavioural information may have been due to the fact that the selected cluster did not contain all of the amplifiers that were affected by the given fault. Although, the use of a 15 minute time offset for generating the fault window guaranteed that at least one cluster within the set of all generated fault clusters did contain the complete set of affected amplifiers, this example illustrates the difficulty involved in manually selecting the appropriate cluster.

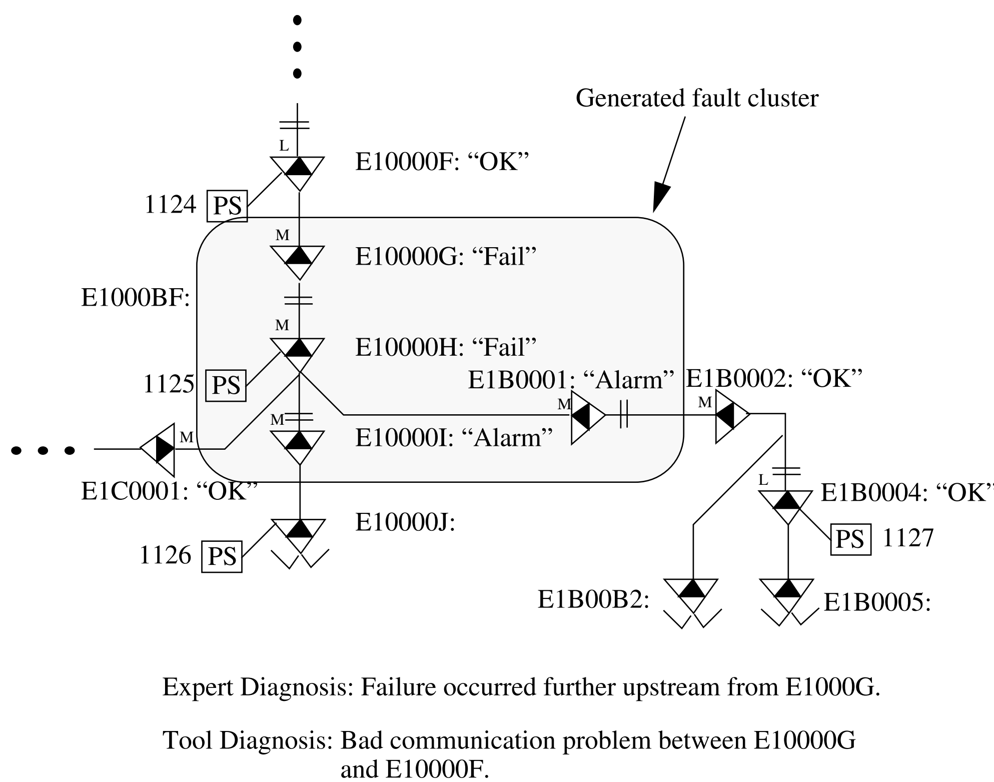
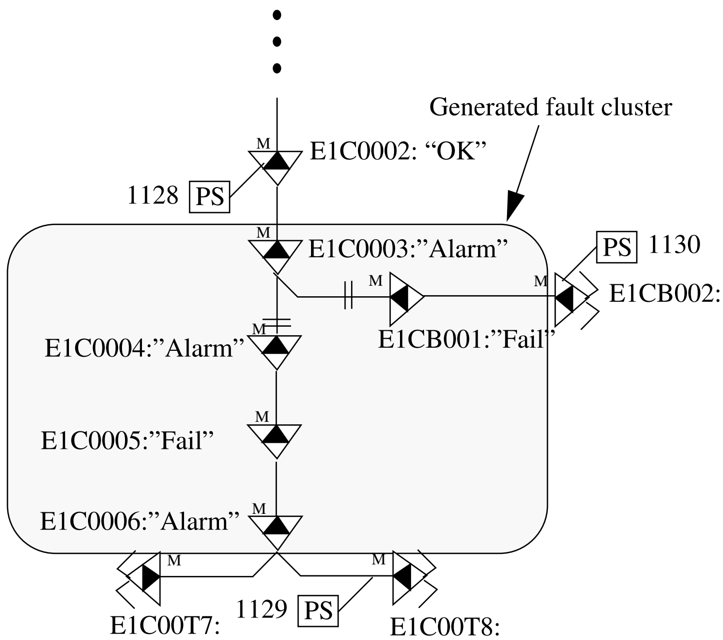


Figure 5.9: Unidentified failure cause fault cluster no.1



Expert Diagnosis: Failure occurred further upstream from E1000G.

Tool Diagnosis: Bad communication problem between E100002 and E100003.

Figure 5.10: Unidentified failure cause fault cluster no.2

In general, as mentioned previously, each network fault will generate a set of fault clusters within the cluster data file, one for each fault message generated by each amplifier affected by the fault. The current prototype tool is not capable of filtering out these additional error messages and, therefore, the appropriate cluster must be selected manually. In part, this problem is due to the fact that the SMT polling software only reports changes in an amplifier's state when the changes occur. Therefore, it is possible that amplifiers upstream of B10000K were actually in error conditions during the time window for which the cluster was generated, but no LOG messages for these amplifiers will be present within the time window in the LOG file. This is beacuse the amplifiers would have reported their error conditions prior to the start of the time window and hence these condtions would have been recored within the LOG file above the area being referenced by the cluster generation software.. If no message for a given amplifier is found within the given section of the LOG files, the current version of the diagnostic tool assumes that the amplifier is operating correctly. This problem could be avoided by using the QA dump data within the cluster server to verify the current state of the given amplifiers during the cluster generation process.

Despite this short coming, the diagnostic tool was able to diagnose the fault as being a result of a bad connection between the head amplifier, B10000K, and its parent's parent, B10000J. This diagnosis does point to a problem with the current bad connection diagnosis in that it will be suggested for any fault cluster for which the diagnostic tool cannot find a cause within the amplifiers included in the cluster. In other words, this diagnostic message will be used by the tool to indicate any fault conditions for which the cluster's fault messages are caused by low signal levels entering the head amplifier from upstream. The fault, therefore, may actually be occurring much further upstream than is stated by the diagnostic message. This problem can be addressed by going to either a real-time diagnostic system capable of filtering out messages from amplifiers within already diagnosed clusters, or by having the cluster server utilize the QA data to verify the operation of the amplifiers instead of relying on just the LOG data. The second method would require that the nominal values for each of the amplifier's also be known. This information was unavailable during the development of the current prototype diagnostic tool.

5.7 Chapter Summary

This chapter presented a comparison of the fault diagnoses obtained from the domain expert and the prototype diagnostic tool for a set of ten network faults. During each comparison, a discussion of the current diagnostic tool's abilities and limitations, in the context of the given type of fault, was also presented. In addition, suggestions for improvements to future versions of the tool were presented. This chapter also illustrated some of the complexities involved in performing fault diagnosis for cable amplifier networks and the methods employed by Rogers' domain expert to accomplish this task. The diagnostic tool was able to produce an accurate diagnosis within the limits of it knowledge domain for all ten of the network faults which were presented to the system. In five cases, these diagnoses were more general than those produced by the domain expert due to the expert's use of information not yet incorporated into the diagnostic tool, such as cable span length. This chapter also demonstrated the need to incorporate additional information, such as the QA dump data and information regrading cable span length, additional network level rules, and additional knowledge bases into future versions of the diagnostic tool. This process is part of the normal development of knowledge based systems in which rapid prototyping is used to develop an initial system and then an upgrading period is employed to improve the tool's performance and functionality.

Chapter 6:

Conclusions

This chapter presents a discussion of the conclusions and recommendations based on the work conducted for this thesis. In addition, suggestions are made regarding future versions of the diagnostic tool.

6.1 Summary of Work

In this thesis, the development of a prototype expert system based diagnostic tool for cable amplifier networks was presented. Specifically the network under consideration was Rogers' Victoria cable plant, but the results obtained from this work are generalizable to any cable network utilizing C-COR's amplifier technology. The amplifiers within this network are continuously polled by status monitoring software located at the network's head end, and this status information is collected in LOG and QA dump data files and transmitted daily to the University of Victoria. These data files, along with Rogers' network description file, were used to develop structural and behavioural models of the network and its operation. An algorithm for creating fault clusters was developed which utilized these models and allowed all the amplifiers affected by a given network fault to be identified. These fault clusters were then loaded into the knowledge base created with the help of a commercial expert system shell, called NEXPERT OBJECT, and diagnosed. This rule base was developed through discussions with the domain expert and utilized a diagnostic tree to isolate the possible causes of the observed fault. This diagnostic tree contains the most common network faults which occur in the Victoria network, as indicated by the LOG and QA dump data files collected over the January to May 1992 time period. Once the prototype system had been developed, it was tested on ten randomly selected network faults exhibiting complex fault clusters from the collected status information. The results of the diagnostic tool were compared to the diagnosis obtained from the same information by the domain expert. For all of the ten fault clusters which were tested in this manner, the diagnostic tool was able to identify the same fault location and cause as that found by the domain expert. Though, in general the domain expert's diagnosis tended to be more specific since they included explanations based on his understanding of the principles of operation of the network components involved in the faults and use of information sources currently not incorporated into the diagnostic tool. The current rule base assumes that each fault cluster is caused by only a single network failure. This assumption is supported by both the domain expert's experience and the collected status data which indicated that multiple failures within the same network neighborhood occur relatively rarely.

This work has demonstrated the proof of concept of using an expert system based diagnostic tool to perform fault diagnoses on cable trunk amplifier networks.

6.2 Future Work

Future versions of the diagnostic tool should be produced by taking a two-pronged approach to address the current tool's limitations. Namely, the clustering algorithm should be tightened to act as a fault message filter and be modified to operate from real-time data. In addition, the knowledge sources identified in chapter 5, such as the QA dump data and cable span length information, should be incorporated into the tool. A corollary step will be the development of additional rule bases, such as one at the amplifier level, to allow the tool to diagnose faults over a larger knowledge domain. In addition, improvements will have to be made to the network level rule base as other limitations in its diagnostic abilities are identified. In addition, the experiential knowledge utilized by the domain expert will need to be incorporated into the diagnostic tool through the development of a shallow reasoning knowledge base to work in conjunction with the deep reasoning structural, and behavioural modeling currently employed. The resulting diagnostic tool will be able to perform a combination of experiential and theoretical reasoning.

If future versions of the tool are to be used to diagnose multi-hubbed cable networks, then modifications to the topology server will have to be made in order to enable it to represent multiple hubs within a single network. In addition large scale use of the tool will require that the data format for the network description file be formalized so that other networks can be represented by the topology server without having to perform any software modifications. The accuracy of any future version of the tool will be limited by the accuracy of the network description file. Any inaccuracies in the file may result in a mis-diagnosis since the tool relies of this file for its view of the network topology which in turn is used both in the generation of the fault clusters and as a key element of the network level diagnostic rule base. Finally, in the future it may be desirable to use an inference engine other than NEXPERT OBJECT to perform the diagnoses due to both inherent limitation as to the size of cluster importable into NEXPERT and due to performance considerations resulting from the future need to produce diagnoses in real-time. Because of how the software comprising the diagnostic tool was developed, no modifications should be required in order to utilize the tool to diagnose faults in other C-COR base trunk cable amplifier networks.

Bibliography

[1] Brooke, J.B. and K.D. Duncan, "Effects of system display format on performance in a fault location task", *Ergonomics*, vol. 24, no. 3, 1981, pp. 175-189.

[2] Rouse, W.B. and S.H. Rouse, "Measures of complexity of fault diagnosis tasks", *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 10, no. 11, 1979, pp. 720-727.

[3] Rouse, W.B., "Problem solving performance of maintenance trainees in fault diagnosis", *Human Factors*, vol. 21, no. 2, 1979, pp. 195-203.

[4] Yoon, W.C. and J.M. Hammer, "Deep-Reasoning Fault Diagnosis: An Aid and a Model", *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 18, no. 4, 1988, pp. 659-676.

[5] Nuckolls, V., "Telecommunications Diagnostic Expert System", *Proceedings of IEEE Global Telecommunications Conference*, Dallas, Texas, Nov. 27-30 1989, pp. 507-511.

[6] Miyazaki, T., H. Fujimoto, M.W. Kim, and M. Wakamoto, "Improving Operation and Maintenance for Switching Networks", *Proceedings of IEEE Global Telecommunications Conference*, Dallas, Texas, Nov. 27-30 1989, pp. 1149-1153.

[7] Tanitomoto, S.L., *The Elements of Artificial Intelligence*, Computer Science Press, Inc., Rockville, Maryland, 1987, p. 207.

[8] Sugawara, T., "A Cooperative LAN Diagnostic and Observation Expert System", *Proceedings of 9th Annual International Phoenix Conference on Computers and Communications*, IEEE, March 1990, pp. 667-674.

[9] Abbott, K.H., "Robust operative diagnosis as problem solving in a hypothesis space", *American Association for Artificial Intelligence Conference*, 1988.

[10] de Klerr, J. and J.S. Brown, "Assumptions and ambiguities in mechanistic mental models", *Mental Models*, Lawrence Erlbaum Association, Hillsdale, NJ, 1983.

[11] Yudkin, R.O., "On Testing Communication Networks", IEEE Journal on Selected Areas in Communications, vol. 6, no. 5, June 1988, pp. 805-812.

[12] C-COR Electronics Inc., *FT-295X Series Feedforward Trunk and Bridgers Stations Instruction Manual*, Rev. 1, Oct. 1988.

[13] C-COR Electronics Inc., *Instruction Manual for Status Monitoring System*, Rev. 0, May 1984.

[14] C-COR Electronics Inc., *Preliminary Instruction Manual for the B-507 Remote Bridger*, Rev. 0, July 1983.

[15] Massey, B., *Introduction to Status Monitoring II*, Rogers Cablesystems Ltd. internal document, 1991.

[16] Massey, B., *Trunk Amplifier Maps*, Rogers Cablesystems Ltd. internal document, 1992.

[17] Massey, B., *personal conversations*, Jan. 1991 - May 1992.

[18] Neuron Data, Inc., *NEXPERT OBJECT: Applications Programming Interface Reference Manual*, Part No. Man-10-700-01, Jan. 1991.

[19] Neuron Data, Inc., *NEXPERT OBJECT: Reference Manual*, Part No. Man-10-400-01,Jan. 1991.

[20] Neuron Data, Inc., *NEXPERT OBJECT: User's Guide*, Part No. Man-10-200-01Jan. 1991.

[21] Neuron Data, Inc., *NEXPERT OBJECT: Functional Description*, Part No. Man-10-300-01, Jan. 1991.

[22] Neuron Data, Inc., *NEXPERT OBJECT: Introduction Manual*, Part No. Man-10-100-01, Jan. 1991.

[23] Neapolitan, R.E., *Probabilistic Reasoning in Expert Systems: Theory and Algorithms*, John Wiley and Sons, Inc., New York, 1990, pp. 372-389.

[24] Pattipati, K.R., *Test Sequencing in Modular Systems*, 1989 IEEE International Conference on Systems, Man, and Cybernetics, Cambridge, Massachusetts, Nov. 14-17, 1989, pp.1221-1223.

[25] Bublin, S.C and R.L. Kashyap, *CONSOLIDATE, Merging /Heuristic Classification with Causal Reasoning in Machine Fault Diagnosis*, Proceedings of the Fourth Conference on Artificial Intelligence Applications, San Diego, California, March 14-18, 1988, pp. 18-25.

[26] Hitson, B.L., *Knowledge-Based Monitoring and Control: An Approach to Understanding the Behavior of TCP/IP Network Protocols*, SIGCOMM 1988 Symposium on Communications Architectures and Protocols, Stanford, California, aug. 16-19, 1988, pp. 210-221.

[27] Sanderson, P.M. and J.M. Murtagh, *Troubleshooting with and Inaccurate Mental Model*, 1989 IEEE International Conference on Systems, Man, and Cybernetics, Cambridge, Massachusetts, Nov. 14-17, 1989, pp.1238-1243.

[28] Marques, T.E., *A Symptom-Driven Expert System for Isolating and Correcting Network Faults*, IEEE Communications, vol. 26, no. 3, March 1988, pp. 6-13.

[29] Chandrasekran, B. and R. Milne, *Reasoning About Structures, Behavior and Function.*, SIGART Newsletter, No. 93, July 1985, pp. 4-7.

[30] Ng, H.T., Model-Based, *Multiple Fault Diagnosis of Time-Varying, Continuous Physical Devices*, Proceedings of the Sixth Conference on Artificial Intelligence Applications, Santa Barbra, California, March 5-9, 1990, pp. 9-15.

[31] Dimopoulos, N.J., K.F. Li, A. Watkins, S. Neville, and A. Rondogainnis, "An Expert Network Analyzer", *Technical Papers: Canadian Cable Television Association 35th Annual Convention*, Vancouver, B.C., May 31 - June 3, 1992, pp. 123-127.

Appendix A:

Topology Server C Routines and Definitions

The following are the listings of the C source code developed to implement the topology server. The code is presented according to the object level it implements.

A.1 Topology Objects Definition File (An_defs.h)

```
#define TRUE  1
#define FALSE 0
#define NEXT          -1
#define NULL  0

#define NAME_STRING_SIZE 8
#define LOC_STRING_SIZE 81
#define TYPE_STRING_SIZE 11
#define PILOT_STRING_SIZE 5
#define AREA_STRING_SIZE 21

typedef intboolean;


typedef struct _Amplifier
        {
         char                          name[NAME_STRING_SIZE];
         char                          location[LOC_STRING_SIZE];
         char                          type[TYPE_STRING_SIZE];
         int                           SMT_num;
         int                           subs;
         char                          pilot[PILOT_STRING_SIZE];
         boolean               termination;
         int                           page_marker;
         struct _Amplifier*next;
         struct _Amplifier*prev;
        } Amplifier;


typedef intExtension_Sub_Network;


typedefstruct _Sub_Network
        {
         Amplifier             *amp;
         struct _Power_Grid*power_grid;
```

```
        struct _Sub_Network_list *branches;
        struct _Sub_Network *prev;
        Extension_Sub_Network*extensions;
        } Sub_Network;


typedefstruct _Sub_Network_list
        {
        Sub_Network           *subnet;
        struct _Sub_Network_list *next;
        struct _Sub_Network_list *prev;
        } Sub_Network_list;


typedefint Power_Grid_type;

typedefstruct _Power_Grid
        {
        Power_Grid_typepower_grid_id;
        Sub_Network_list*subnet_list;
        struct _Power_Grid*next;
        struct _Power_Grid*prev;
        } Power_Grid;


typedef intExtension_Network;

typedef struct
        {
        Sub_Network_list*trunk_list;
        Sub_Network_list*subnet_list;
        Power_Grid   *power_grid_list;
        Amplifier      *amp_list;
        char                      area[AREA_STRING_SIZE];
        Extension_Network*extensions;
        } Network;
```

A.2 Network Level Routines (AN_Network.c)

```
#include <strings.h>
#include <stdio.h>
#include “AN_defs.h”
#include “AN_Sub_Network.h”
#include “AN_Sub_Network_list.h”
#include “AN_Amplifier.h”
#include “AN_Power_Grid.h”
```

```
/****************************************************************
 * *
 * Network Object Routines. *
 * *
 * *
 ****************************************************************/


Network*create_network()
/****************************************************************
 * Description: *
 *       This procedure creates a network data structure. *
 * Error Handling: *
 * None. *
 * Returns: *
 *       Pointer to the created network. *
 ****************************************************************/
{
        Network *temp;

        temp = (Network *) malloc (sizeof(Network));
        temp->trunk_list = NULL;
        temp->subnet_list = NULL;
        temp->power_grid_list= NULL;
        temp->amp_list= NULL;
        strcpy(temp->area,"");
        temp->extensions = NULL;

        return (temp);
}


void set_subnet_list_network(subnet_list,net)
Sub_Network_list*subnet_list;
Network                **net;
/***************************************************************
 * Description: *
 *       This procedure sets the subnet field of the network. *
 * Error Handling: *
 * None. *
 * Returns: *
 * None. *
 ***************************************************************/
{
        (*net)->subnet_list = subnet_list;
}


Sub_Network_list *get_subnet_list_network(net)
Network**net;
/***************************************************************
```

```
 * Description: *
 *       This procedure gets the subnet field of the network. *
 * Error Handling: *
 * None. *
 * Returns: *
 *       Value of the subnet field of the network. *
 **************************************************************/
{
        if (*net == NULL) return (NULL);
        return((*net)->subnet_list);
}


void set_area_network(area,net)
char            *area;
Network         **net;
/**************************************************************
 * Description: *
 *       This procedure sets the area field of the network. *
 * Error Handling: *
 * None. *
 * Returns: *
 * None. *
 **************************************************************/
{
        if (area == NULL) area = "";
        strcpy((*net)->area,area);
}


char *get_area_network(net)
Network**net;
/**************************************************************
 * Description: *
 *       This procedure get a pointer to the area field of the network. *
 * Error Handling: *
 * None. *
 * Returns: *
 *       Pointer to the area. *
 **************************************************************/
{
        if (*net == NULL) return (NULL);
        return((*net)->area);
}


Power_Grid *get_power_grid_list_network(net)
Network **net;
{
        if (*net == NULL) return (NULL);
        return ((*net)->power_grid_list);
```

```
}


void set_power_grid_list_network(power_grid, net)
Power_Grid *power_grid;
Network **net;
{
	(*net)->power_grid_list = power_grid;
}


Amplifier *get_amp_list_network(net)
Network **net;
{
	if (*net == NULL) return (NULL);
	return ((*net)->amp_list);
}


void set_amp_list_network(amp_list, net)
Amplifier*amp_list;
Network **net;
{
	(*net)->amp_list = amp_list;
}


Sub_Network_list *get_trunk_list_network(net)
Network**net;
{
	if (*net == NULL) return (NULL);
	return((*net)->trunk_list);
}


void set_trunk_list_network(trunk_list,net)
Sub_Network_list*trunk_list;
Network			**net;
{
	(*net)->trunk_list = trunk_list;
}


Sub_Network_list *next_trunk_list_network(trunk_list)
Sub_Network_list**trunk_list;
{
	if (*trunk_list == NULL) return (NULL);
	return(next_sub_network_list(trunk_list));
}
```

```
Sub_Network_list *prev_trunk_list_network(trunk_list)
Sub_Network_list**trunk_list;
{
        if (*trunk_list == NULL) return (NULL);
        return(prev_sub_network_list(trunk_list));
}



Sub_Network_list *add_and_set_trunk_list_network(subnet, net)
Sub_Network*subnet;
Network        **net;
{
        Sub_Network_list *i;
        Sub_Network_list*temp;

        i = get_trunk_list_network(net);
        if   (i   !=   NULL)   for   (i=get_trunk_list_network(net);   i->next   !=   NULL;
i=next_trunk_list_network(&i));

        temp = create_and_set_all_sub_network_list(subnet,&i);

        if (get_trunk_list_network(net) == NULL) set_trunk_list_network(temp,net);

        return (temp);
}


void make_trunk_list_network(net)
Network**net;
{
        Sub_Network_list *trunk_list;
        Sub_Network_list *subnet_list,*i;
        Sub_Network  *subnet,*prev;

        subnet_list = get_subnet_list_network(net);
        trunk_list = get_trunk_list_network(net);

        for (i=get_subnet_list_network(net); i!= NULL; i = next_sub_network_list(&i))
        {
                subnet = get_subnet_sub_network_list(&i);
                prev = get_prev_subnet(&subnet);
                if (prev == NULL) add_and_set_trunk_list_network(subnet,net);
        }
}


void del_trunk_list_network(trunk_list, net)
Sub_Network_list*trunk_list;
Network                 **net;
{
```

```
	Sub_Network_list*subnet_list;

	subnet_list = get_trunk_list_network(net);
	del_sub_network_list(trunk_list,&subnet_list);
	set_trunk_list_network(subnet_list,net);
}


void destroy_trunk_list_network (net)
Network**net;
{
	Sub_Network_list*i;

	for (i=get_trunk_list_network(net); i!=NULL; i=next_trunk_list_network(&i))
		del_trunk_list_network(i,net);
}


Sub_Network_list *find_trunk_list_network(subnet,net)
Sub_Network*subnet;
Network		**net;
{
	Sub_Network_list*i;

	for (i = get_trunk_list_network(net);
	 ((i != NULL) && (get_subnet_sub_network_list(&i) != subnet));
	 i = next_trunk_list_network(&i));

	return(i);
}


void destroy_network(net)
Network**net;
/**************************************************************
 * Description: *
 *	This procedure deletes the network net. *
 *	The deleted network is then freed. *
 * Error Handling: *
 * None. *
 * Returns: *
 *	None. *
 **************************************************************/
{

	/* destroy_subnet_list(net); NYI */
	destroy_power_grid(net);
	destroy_amp_list(net);
	destroy_trunk_list_network(net);
	free(*net);
	*net = NULL;
```

```
}


void write_network (filename, net)
char     *filename;
Network **net;
{
        FILE                    *file_id;

        Sub_Network  *subnet,*parent_subnet;
        Sub_Network_list*i;
        Sub_Network_list*branch;
        Power_Grid    *power_grid;
        Amplifier       *amp,*parent_amp;

        int                     j;

        char                    *amp_name;
        char                    *location;
        char                    *type;
        int                     SMT_num;
        int                     subs;
        char                    *pilot;
        boolean                 termination;
        int                     page;
        int                     power_grid_id;
        char                    *branch1;
        char                    *branch2;
        char                    *branch3;
        char                    *branch4;
        char                    *parent;

        char                    *temp;
        char                    termination_char;

        file_id = fopen(filename, "w");

        for (i= get_subnet_list_network(net); i != NULL; i = next_sub_network_list(&i))
        {
                branch1 = "";
                branch2 = "";
                branch3 = "";
                branch4 = "";

                subnet          = get_subnet_sub_network_list(&i);

                parent_subnet = get_prev_subnet(&subnet);
                parent_amp  = get_amp_subnet(&parent_subnet);
                parent            = get_name_amp(&parent_amp);
                if (parent == NULL) parent = "";
```

```
power_grid = get_power_grid_subnet(&subnet);
power_grid_id= get_power_grid_id_power_grid(&power_grid);

amp          = get_amp_subnet(&subnet);
amp_name = get_name_amp(&amp);
location= get_location_amp(&amp);
type         = get_type_amp(&amp);
SMT_num = get_SMT_num_amp(&amp);
subs    = get_subs_amp(&amp);
pilot   = get_pilot_amp(&amp);
termination = get_termination_amp(&amp);
page         = get_page_marker_amp(&amp);

termination_char = ‘F’;
if (termination == TRUE) termination_char = ‘T’;
if (termination == NEXT) termination_char = ‘N’;


branch = get_branches_subnet(&subnet);

for (j=1; ((j <= 4) && (branch != NULL)); j++)
{

        subnet = get_subnet_sub_network_list(&branch);
        amp = get_amp_subnet(&subnet);
        temp = get_name_amp(&amp);

        if (j == 1) branch1 = temp;
        if (j == 2) branch2 = temp;
        if (j == 3) branch3 = temp;
        if (j == 4) branch4 = temp;

        branch = next_sub_network_list(&branch);
}

if (strcmp(location,””) == 0) location = “NULL”;
if (strcmp(type,””) == 0) type = “NULL”;
if (strcmp(pilot,””) == 0) pilot = “NULL”;
if (strcmp(branch1,””) == 0) branch1 = “NULL”;
if (strcmp(branch2,””) == 0) branch2 = “NULL”;
if (strcmp(branch3,””) == 0) branch3 = “NULL”;
if (strcmp(branch4,””) == 0) branch4 = “NULL”;
if (strcmp(parent,””) == 0) parent = “NULL”;


fprintf(file_id,”%s \”%s\” %d %d %s %c %d %d %s %s %s %s %s %s\n”,
        amp_name, location, SMT_num, subs, pilot, termination_char,
        power_grid_id, page, branch1, branch2, branch3, branch4, type,
parent);
    }
```

```
        fclose(file_id);
}


Network *read_network(file_name)
char    *file_name;
{

        Network*net;
 FILE *file_id;
        int             nd;

 int            i;
        int     j;
        char    tmp_char;
        int             termination_int;

 char   amp_name[NAME_STRING_SIZE];
 char   location[LOC_STRING_SIZE];
 char   type[TYPE_STRING_SIZE];
 int    SMT_num;
 int    subs;
 char   pilot[PILOT_STRING_SIZE];
 char   termination[2];
 int    page;
 int    power_grid_id;
 char   branch1[NAME_STRING_SIZE];
 char   branch2[NAME_STRING_SIZE];
 char   branch3[NAME_STRING_SIZE];
 char   branch4[NAME_STRING_SIZE];
 char   parent[NAME_STRING_SIZE];

        net = create_network();

 file_id = fopen(file_name,"r");
 if (file_id != NULL)
 {

 nd = fscanf(file_id,"%s",amp_name);

 while (nd != EOF)
 {
 tmp_char = fgetc(file_id);

 for (j = 0; tmp_char != '"';j++)
 {
 tmp_char = fgetc(file_id);

 if (j > 15) exit(0);
 }
```

```
 tmp_char = getc(file_id);
 for (i = 0; tmp_char != '\"'; i++)
 {
 location[i] = tmp_char;
 tmp_char = getc(file_id);
 }
 location[i] = '\0';

 fscanf(file_id,"%d %d %s %s %d %d %s %s %s %s %s %s\n",
 &SMT_num, &subs, pilot, termination, &power_grid_id, &page,
 branch1, branch2, branch3, branch4, type, parent);

 if (strcmp(branch1,"NULL") == 0) branch1[0] = NULL;
 if (strcmp(branch2,"NULL") == 0) branch2[0] = NULL;
 if (strcmp(branch3,"NULL") == 0) branch3[0] = NULL;
 if (strcmp(branch4,"NULL") == 0) branch4[0] = NULL;
 if (strcmp(location,"NULL") == 0) location[0] = NULL;
 if (strcmp(amp_name,"NULL") == 0) amp_name[0] = NULL;
 if (strcmp(termination,"NULL") == 0) termination[0] = NULL;
 if (strcmp(pilot,"NULL") == 0) pilot[0] = NULL;
 if (strcmp(type,"NULL") == 0) type[0] = NULL;
 if (strcmp(parent,"NULL") == 0) parent[0] = NULL;

                                termination_int = FALSE;
                                if ((strcmp(termination,"T") == 0) || (strcmp(termination,"t") == 0))
termination_int = TRUE;
                                if (strcmp(termination,"N") == 0) termination_int = NEXT;

                                if (strcmp(location,"") != 0)
                                {

add_and_set_all_subnet(amp_name,location,type,SMT_num,subs,pilot,termination_int,p
age,power_grid_id,

branch1,branch2,branch3,branch4,parent,&net);
                                }

                                nd = fscanf(file_id,"%s",amp_name);
                        }
            }

            fclose(file_id);

            make_trunk_list_network(&net);

            return(net);
}
```

A.3 Sub Network Level Routines (AN_Sub_Network.c)

```
#include <stdio.h>
#include <strings.h>
#include "AN_defs.h"
#include "AN_Network.h"
#include "AN_Amplifier.h"
#include "AN_Sub_Network_list.h"
#include "AN_Power_Grid.h"

/*****************************************************************
 * *
 * Subsubnet Object Routines. *
 * *
 * Note: Extensions no implemented yet. *
 * *
 *****************************************************************/


Sub_Network *create_subnet()
/*****************************************************************
 * Description: *
 *      This procesure allocates the memory for a Sub_Network data structure. *
 * Error Handling: *
 * None. *
 * Returns: *
 *      A pointer to the subsubnet. *
 *****************************************************************/
{
        Sub_Network*temp;

        temp = (Sub_Network *) malloc (sizeof (Sub_Network));

        temp->amp       = NULL;
        temp->power_grid = NULL;
        temp->branches = NULL;
        temp->prev      = NULL;
        temp->extensions = NULL;

        return(temp);
}


void destroy_subnet(subnet)
Sub_Network**subnet;
/*****************************************************************
 * Description: *
 *      This procesure releases the memory allocated to the subsubnet *
 *      and all the subsubnets pointed to by it. *
 * Error Handling: *
 * None. *
```

```
 * Returns: *
 * None. *
 *****************************************************************/
{
        free((*subnet)->amp);
        free((*subnet)->prev);
        free((*subnet)->extensions);
        free(*subnet);
        *subnet = NULL;
}



Amplifier *get_amp_subnet(subnet)
Sub_Network**subnet;
/*****************************************************************
 * Description: *
 *      This procedure returns a pointer to the amplifier for a specified *
 *      subsubnet. *
 * Error Handling: *
 * None. *
 * Returns: *
 *      A pointer to the amplifier. *
 *****************************************************************/
{
        if (*subnet == NULL) return (NULL);
        return((*subnet)->amp);
}



void set_amp_subnet(amp,subnet)
Amplifier*amp;
Sub_Network**subnet;
/*****************************************************************
 * Description: *
 *      This procedure set the amplifier field to a particular amplifier *
 *      for the specified subsubnet. *
 * Error Handling: *
 * None. *
 * Returns: *
 *      None. *
 *****************************************************************/
{
        (*subnet)->amp = amp ;
}


Power_Grid *get_power_grid_subnet(subnet)
Sub_Network**subnet;
/*****************************************************************
```

```
 * Description: *
 *       This procedure returns a pointer to the power grid for a specified *
 *       subsubnet. *
 * Error Handling: *
 * None. *
 * Returns: *
 *       A pointer to the power grid. *
 ***************************************************************/
{
        if (*subnet == NULL) return (NULL);
        return((*subnet)->power_grid);
}


void set_power_grid_subnet(power_grid,subnet)
Power_Grid*power_grid;
Sub_Network**subnet;
/***************************************************************
 * Description: *
 *       This procedure set the power_grid field to a particular power_grid *
 *       for the specified subsubnet. *
 * Error Handling: *
 * None. *
 * Returns: *
 *       None. *
 ***************************************************************/
{
        (*subnet)->power_grid = power_grid;
}


Sub_Network_list *get_branches_subnet(subnet)
Sub_Network**subnet;
/***************************************************************
 * Description: *
 *       This procedure returns a pointer to the branches of a specified *
 *       subsubnet. *
 * Error Handling: *
 * None. *
 * Returns: *
 *       A pointer to the list of branches. *
 ***************************************************************/
{
        if (*subnet == NULL) return (NULL);
        return((*subnet)->branches);
}


void set_branches_subnet(branches,subnet)
Sub_Network_list*branches;
Sub_Network  **subnet;
```

```
/************************************************************
 * Description: *
 *       This procedure set the branches to a particular branch list pointer *
 *       for the specified subsubnet. *
 * Error Handling: *
 * None. *
 * Returns: *
 *       None. *
 *************************************************************/
{
        (*subnet)->branches = branches;
}


Sub_Network *get_prev_subnet(subnet)
Sub_Network**subnet;
/************************************************************
 * Description: *
 *       This procedure returns a pointer to the previous subsubnet. *
 * Error Handling: *
 * None. *
 * Returns: *
 *       A pointer to the previous subsubnet. *
 *************************************************************/
{
        if (*subnet == NULL) return (NULL);
        return((*subnet)->prev);
}


void set_prev_subnet(prev,subnet)
Sub_Network*prev;
Sub_Network**subnet;
/************************************************************
 * Description: *
 *       This procedure sets the previous subsubnet *
 *       for the specified subsubnet. *
 * Error Handling: *
 * None. *
 * Returns: *
 *       None. *
 *************************************************************/
{
        (*subnet)->prev = prev;
}


Sub_Network *add_branch_subnet (add_subnet,subnet)
Sub_Network *add_subnet;
Sub_Network **subnet;
/************************************************************
```

```
 * Description: *
 *       This procedure adds a subnet to the branches of a given subnet and *
 *       returns a pointer to this added subnet. *
 * Error Handling: *
 * None. *
 * Returns: *
 *       None. *
 ****************************************************************/
{
        Sub_Network_list*temp;

        temp = add_and_create_sub_network_list(&((*subnet)->branches));

        set_subnet_sub_network_list(add_subnet,&temp);
        set_prev_subnet(*subnet,&add_subnet);

        return (add_subnet);
}


void del_subnet (subnet,net)
Sub_Network **subnet;
Network **net;
/****************************************************************
 * Description: *
 *       This procedure deletes a subnet from the data structure and also *
 *       removes all of its children from the structure. *
 * Error Handling: *
 * None. *
 * Returns: *
 *       None. *
 ****************************************************************/
{
        /*
        */
}


Sub_Network *find_amp_subnet(amp_name,subnet)
char            *amp_name;
Sub_Network**subnet;
{
        Sub_Network_list*i;
        Sub_Network  *temp,*temp2;
        Amplifier       *amp;

        temp = NULL;
        temp2 = NULL;

        if ((*subnet != NULL) && (amp_name != NULL))
        {
```

```
		if (get_amp_subnet(subnet) != NULL)
		{
			amp = get_amp_subnet(subnet);
			/*		fprintf(stdout,"amp		=		%s
%d\n",get_name_amp(&amp),strcmp(get_name_amp(&amp),amp_name)); */
			if (strcmp(get_name_amp(&amp),amp_name) != 0)
			{
				if (get_branches_subnet(subnet) != NULL)
				{
					for (i=get_branches_subnet(subnet); i != NULL;
i=next_sub_network_list(&i))
					{
						temp = get_subnet_sub_network_list(&i);
						temp2					=
find_amp_subnet(amp_name,&temp);
						if (temp2 != NULL) break;
					}
				}
			}
			else temp2 = *subnet;
		}
	}
	return(temp2);
}

/****************************************************************
 *								*
 *		Routines to deal with the subnet list at net->subnet_list *
 *								*
 ****************************************************************/


Sub_Network_list *next_subnet_list(subnet_list)
Sub_Network_list**subnet_list;
{
	if (*subnet_list == NULL) return (NULL);
	return(next_sub_network_list(subnet_list));
}


Sub_Network_list *prev_subnet_list(subnet_list)
Sub_Network_list**subnet_list;
{
	if (*subnet_list == NULL) return (NULL);
	return(prev_sub_network_list(subnet_list));
}



Sub_Network_list *add_subnet_subnet_list(subnet, net)
Sub_Network*subnet;
```

```
Network         **net;
{
        Sub_Network_list *i;
        Sub_Network_list*temp;

        i = get_subnet_list_network(net);
        if (i != NULL) for (i=get_subnet_list_network(net); next_subnet_list(&i) !=
NULL; i=next_subnet_list(&i));

        temp = create_and_set_all_sub_network_list(subnet,&i);

        if (get_subnet_list_network(net) == NULL) set_subnet_list_network(temp,net);

        return (temp);
}


Sub_Network_list *add_and_create_subnet_subnet_list(net)
Network**net;
{
        Sub_Network*temp;

        temp = create_subnet();

        return(add_subnet_subnet_list(temp,net));
}



void del_subnet_list(del_subnet, net)
Sub_Network_list*del_subnet;
Network                 **net;
{
        Sub_Network_list*subnet_list;

        subnet_list = get_subnet_list_network(net);
        del_sub_network_list(del_subnet,&subnet_list);
        set_subnet_list_network(subnet_list,net);
}


void destory_subnet_list (net)
Network**net;
{
        Sub_Network_list*i;

        for (i=get_subnet_list_network(net); i!=NULL; i=next_subnet_list(&i))
                del_subnet_list(i,net);
}
```

```
Sub_Network_list *find_subnet_list(subnet,net)
Sub_Network*subnet;
Network         **net;
{
        Sub_Network_list*i;

        for (i = get_subnet_list_network(net);
         ((i != NULL) && (get_subnet_sub_network_list(&i) != subnet));
         i = next_subnet_list(&i));

        return(i);
}



Sub_Network_list *find_amp_name_subnet_list(name,net)
char    *name;
Network**net;
{
        Sub_Network  *subnet;
        Amplifier       *amp;
        char                    *amp_name;
        Sub_Network_list*i;

        for ( i=get_subnet_list_network(net); (i != NULL); i=next_sub_network_list(&i))
        {
                subnet = get_subnet_sub_network_list(&i);
                amp = get_amp_subnet(&subnet);
                amp_name = get_name_amp(&amp);

                if (strcmp(amp_name,name) == 0) return (i);
        }

        return (i);
}


/*****************************************************************/


Sub_Network
*add_and_set_all_subnet(name,location,type,SMT_num,subs,pilot,termination,page,

power_grid_id,branch1,branch2,branch3,branch4,parent,net)
char            *name;
char            *location;
char            *type;
int             SMT_num;
int             subs;
```

```
char            *pilot;
boolean         termination;
int             page;
int             power_grid_id;
char            *branch1;
char            *branch2;
char            *branch3;
char            *branch4;
char            *parent;
Network         **net;
{
        int                     i;
        char                    *temp;
        Amplifier       *amp;
        Sub_Network  *subnet,*branch_subnet,*parent_subnet;
        Sub_Network_list*temp_subnet_list;
        Power_Grid      *power_grid;

        amp                                                                                          =
create_and_set_all_amp(name,location,type,SMT_num,subs,pilot,termination,page,net);
        temp_subnet_list = find_amp_name_subnet_list(name,net);
        /* fprintf(stdout,"Find amp %d\n",temp_subnet_list); */
        if (temp_subnet_list == NULL)
        {
                temp_subnet_list = add_and_create_subnet_subnet_list(net);
                /* fprintf(stdout,"Find Amp FAILED\n"); */
        }
        subnet = get_subnet_sub_network_list(&temp_subnet_list);
        set_amp_subnet(amp,&subnet);
        /* fprintf(stdout,"Created amp %s\n",get_name_amp(&amp)); */

        if (power_grid_id != NULL)
        {
                power_grid = create_and_set_all_power_grid(power_grid_id, subnet, net);
                set_power_grid_subnet(power_grid,&subnet);
        }

        for (i=1; i<=4; i++)
        {
                if (i == 1) temp = branch1;
                if (i == 2) temp = branch2;
                if (i == 3) temp = branch3;
                if (i == 4) temp = branch4;

                if ((temp != NULL) && strcmp(temp,"") != 0)
                {
                /* fprintf(stdout,"Adding Branches %s %s\n",name,temp); */
                        temp_subnet_list = find_amp_name_subnet_list(temp,net);
                        if (temp_subnet_list == NULL)
                        {
```

```
                                        temp_subnet_list                                    =
add_and_create_subnet_subnet_list(net);
                                        /* fprintf(stdout,"BRANCHES Find Amp FAILED\n"); */
                                        branch_subnet                                       =
get_subnet_sub_network_list(&temp_subnet_list);

set_amp_subnet(create_and_set_all_amp(temp,NULL,NULL,NULL,NULL,NULL,NUL
L,NULL,net),&branch_subnet);
                                }
                                branch_subnet                                               =
get_subnet_sub_network_list(&temp_subnet_list);
                                add_branch_subnet(branch_subnet,&subnet);
                                /* set_prev_subnet(subnet,&branch_subnet); */
                                /* fprintf(stdout,"BRANCHES Created amp %s\n",temp); */
                        }
                }

                if (strcmp(parent, "") != 0)
                {
                        temp_subnet_list = find_amp_name_subnet_list(parent,net);
                        if (temp_subnet_list == NULL)
                        {
                                temp_subnet_list = add_and_create_subnet_subnet_list(net);
                                parent_subnet = get_subnet_sub_network_list(&temp_subnet_list);

set_amp_subnet(create_and_set_all_amp(parent,NULL,NULL,NULL,NULL,NULL,NU
LL,NULL,net),&parent_subnet);
                        }
                        parent_subnet = get_subnet_sub_network_list(&temp_subnet_list);
                        set_prev_subnet(parent_subnet,&subnet);
                }

                return (subnet);

}
```

A.4 Power Grid Level Routines (AN_Power_Grid.c)

```
#include <strings.h>
#include "AN_defs.h"
#include "AN_Network.h"
#include "AN_Sub_Network_list.h"


/**************************************************************
 * *
 * Power Grid list Routines. *
 * *
 **************************************************************/
```

```
Power_Grid *create_power_grid()
{
/***************************************************************
 * Description: *
 *       This procedure allocates the memory used by the power_grid object. *
 * Error Handling: *
 * None. *
 * Returns: *
 * Pointer to the power grid object. *
 ***************************************************************/
         Power_Grid*temp;

         temp = (Power_Grid *) malloc (sizeof (Power_Grid));

         temp->power_grid_id = NULL;
         temp->subnet_list = NULL;
         temp->next = NULL;
         temp->prev = NULL;

         return (temp);
}


Power_Grid_typeget_power_grid_id_power_grid(power_grid)
Power_Grid**power_grid;
{
         if (*power_grid == NULL) return (NULL);
         return ((*power_grid)->power_grid_id);
}


void set_power_grid_id_power_grid(power_grid_id,power_grid)
Power_Grid_typepower_grid_id;
Power_Grid**power_grid;
{
         (*power_grid)->power_grid_id = power_grid_id;
}


Sub_Network_list *get_subnet_list_power_grid (power_grid)
Power_Grid**power_grid;
/***************************************************************
 * Description: *
 *       This procedure gets a pointer to a subnet_list stored in the Power *
 *       grid list. *
 * Error Handling: *
 * None. *
 * Returns: *
 * pointer to the subnetwork. *
```

```
*****************************************************************/
{
        if (*power_grid == NULL) return (NULL);
        return((*power_grid)->subnet_list);
}

void *set_subnet_list_power_grid (subnet_list, power_grid)
Sub_Network_list *subnet_list;
Power_Grid **power_grid;
/*****************************************************************
 * Description: *
 *      This procedure sets the subnet list field in the power grid list. *
 * Error Handling: *
 * None. *
 * Returns: *
 * None. *
 *****************************************************************/
{
        (*power_grid)->subnet_list = subnet_list;
}


Power_Grid *next_power_grid (power_grid)
Power_Grid**power_grid;
/*****************************************************************
 * Description: *
 *      This procedure returns a pointer to the next element in the power *
 *      grid list. *
 * Error Handling: *
 * None. *
 * Returns: *
 * pointer to next element in the power grid list. *
 *****************************************************************/
{
        if (*power_grid == NULL) return (NULL);
        return((*power_grid)->next);
}


Power_Grid *prev_power_grid (power_grid)
Power_Grid**power_grid;
/*****************************************************************
 * Description: *
 *      This procedure returns a pointer to the prev element in the power *
 *      grid list. *
 * Error Handling: *
 * None. *
 * Returns: *
 * pointer to prev element in the power grid list. *
 *****************************************************************/
{
```

```
        if (*power_grid == NULL) return (NULL);
        return((*power_grid)->prev);
}


Power_Grid *add_power_grid(add_grid,net)
Power_Grid *add_grid;
Network **net;
{
        Power_Grid *power_grid;

        power_grid = get_power_grid_list_network(net);

        if (power_grid == NULL)
        {
                set_power_grid_list_network(add_grid,net);
        }
        else
        {
                add_grid->next = power_grid->next;
                power_grid->next = add_grid;
                add_grid->prev= power_grid;
        }

        return (add_grid);
}


Power_Grid *add_and_create_power_grid (net)
Network         **net;
{
        Power_Grid *temp;

        temp = create_power_grid();
        return( add_power_grid (temp, net));
}


void del_power_grid (del_grid, net)
Power_Grid *del_grid;
Network         **net;
{
        Power_Grid*power_grid_list;

        power_grid_list = get_power_grid_list_network(net);

 if ((del_grid != NULL) && (power_grid_list != NULL))
 {
 if (power_grid_list == del_grid)
 {
 set_power_grid_list_network(del_grid->next,net);
```

```
 if (del_grid->next != NULL) (del_grid->next)->prev = NULL;
 }
 else
 {
 if (del_grid->prev != NULL) (del_grid->prev)->next = del_grid->next;

 if (del_grid->next != NULL) (del_grid->next)->prev = del_grid->prev;
 }
 }

 free (del_grid);
 del_grid = NULL;

}


void destroy_power_grid (net)
Network**net;
/****************************************************************
 * Description: *
 *       This procedure destory the power grid list. *
 * Error Handling: *
 * None. *
 * Returns: *
 * None. *
 ****************************************************************/
{
 Power_Grid *i;

 if (get_power_grid_list_network(net) != NULL)
 {
   for   (i=get_power_grid_list_network(net);   i   !=   NULL;   i=next_power_grid(&i))
del_power_grid(i,net);
 }

}


Power_Grid *find_power_grid_id_power_grid(id,net)
Power_Grid_type id;
Network        **net;
/****************************************************************
 * Description: *
 * This procedure finds the power grid with the power grid id of id. *
 * Error Handling: *
 * None. *
 * Returns: *
 * Pointer to the power grid. *
 ****************************************************************/
{
 Power_Grid *i;
```

```
        for         (i=get_power_grid_list_network(net);         ((i!=NULL)         &&
(get_power_grid_id_power_grid(&i) != id)); i=next_power_grid(&i));

 return(i);
}


Power_Grid *find_subnet_list_power_grid(subnet_list,net)
Sub_Network_list *subnet_list;
Network                  **net;
/*************************************************************
 * Description: *
 * This procedure finds the power grid with the specified subnet_list. *
 * Error Handling: *
 * None. *
 * Returns: *
 * Pointer to the power grid. *
 *************************************************************/
{
 Power_Grid *i;

 for (i=get_power_grid_list_network(net);
         ((i!=NULL) && (get_subnet_list_power_grid(&i) != subnet_list));
         i=next_power_grid(&i));

 return(i);
}


Power_Grid *create_and_set_all_power_grid(id, subnet, net)
int              id;
Sub_Network*subnet;
Network          **net;
{
        Power_Grid    *temp;
        Sub_Network_list *subnet_list,*temp2;

        temp = find_power_grid_id_power_grid(id,net);
        if (temp == NULL)
        {
                temp = add_and_create_power_grid(net);
                set_power_grid_id_power_grid(id,&temp);
        }

        subnet_list = get_subnet_list_power_grid(&temp);

        temp2 = create_and_set_all_sub_network_list(subnet,&subnet_list);

        set_subnet_list_power_grid(subnet_list,&temp);
```

```
        return (temp);
}
```

A.5 Amplifier Level Routines (AN_Amplifier.c)

```
#include <strings.h>
#include <stdio.h>
#include “AN_defs.h”
#include “AN_Network.h”

/****************************************************************
 * *
 * Amplifier Object Routines. *
 * *
 ****************************************************************/




Amplifier *create_amp()
/****************************************************************
 * Description: *
 *       This procedure allocates memory for a amplifier data structure. *
 * Error Handling: *
 * None. *
 * Returns: *
 * Pointer to the created data structure. *
 ****************************************************************/
{
        Amplifier*temp;

        temp = (Amplifier *) malloc (sizeof (Amplifier));
        strcpy(temp->name,”XXX”);
        strcpy(temp->location,””);
        strcpy(temp->type,””);
        temp->SMT_num = 0;
        temp->subs = 0;
        strcpy(temp->pilot,””);
        temp->termination = 0;
        temp->page_marker = 0;
        temp->next = NULL;
        temp->prev = NULL;
        return(temp);
}


char    *get_name_amp(amp)
Amplifier**amp;
/****************************************************************
```

```
 * Description: *
 *       This procedure gets the name field of the amplifier specified by amp. *
 * Error Handling: *
 * None. *
 * Returns: *
 *       Pointer to the name of the amp. *
 ***************************************************************/
{
        if (*amp == NULL) return (NULL);
        return((*amp)->name);
}


void    set_name_amp(name,amp)
char            *name;
Amplifier**amp;
/***************************************************************
 * Description: *
 *       This procedure sets the name field of the amplifier specified by amp. *
 * Error Handling: *
 * None. *
 * Returns: *
 * None. *
 ***************************************************************/
{
        if (name == NULL) name = ““;
        strcpy((*amp)->name,name);
}


char *get_location_amp(amp)
/***************************************************************
 * Description: *
 *       This procedure gets the location field of the amplifier *
 * specified by amp.                              *
 * Error Handling: *
 * None. *
 * Returns: *
 * None. *
 ***************************************************************/
Amplifier **amp;
{
        if (*amp == NULL) return (NULL);
        return((*amp)->location);
}


void set_location_amp(location,amp)
/***************************************************************
 * Description: *
 *       This procedure sets the location field of the amplifier *
```

```
 * specified by amp.                          *
 * Error Handling: *
 * None. *
 * Returns: *
 * None. *
 **************************************************************/
char            *location;
Amplifier **amp;
{
        if (location == NULL) location = "";
        strcpy((*amp)->location,location);
}


char *get_type_amp(amp)
Amplifier **amp;
{
        if (*amp == NULL) return (NULL);
        return((*amp)->type);
}


void set_type_amp(type,amp)
char            *type;
Amplifier **amp;
{
        if (type == NULL) type = "";
        strcpy((*amp)->type,type);
}


int get_SMT_num_amp(amp)
Amplifier **amp;
{
        if (*amp == NULL) return (NULL);
        return((*amp)->SMT_num);
}


void set_SMT_num_amp(num,amp)
int             num;
Amplifier **amp;
{
        (*amp)->SMT_num = num;
}


int get_subs_amp(amp)
Amplifier **amp;
{
        if (*amp == NULL) return (NULL);
```

```
	return((*amp)->subs);
}


void set_subs_amp(num,amp)
int		num;
Amplifier **amp;
{
	(*amp)->subs = num;
}



char *get_pilot_amp(amp)
Amplifier **amp;
{
	if (*amp == NULL) return (NULL);
	return((*amp)->pilot);
}


void set_pilot_amp(pilot,amp)
char		*pilot;
Amplifier **amp;
{
	if (pilot == NULL) pilot = ““;
	strcpy((*amp)->pilot,pilot);
}


boolean get_termination_amp(amp)
Amplifier**amp;
{
	if (*amp == NULL) return (NULL);
	return((*amp)->termination);
}


void set_termination_amp(termination,amp)
boolean		termination;
Amplifier**amp;
{
	(*amp)->termination = termination;
}


int get_page_marker_amp(amp)
Amplifier**amp;
{
	if (*amp == NULL) return (NULL);
	return((*amp)->page_marker);
```

```
}


void set_page_marker_amp(page,amp)
int             page;
Amplifier**amp;
{
        (*amp)->page_marker = page;
}


Amplifier *next_amp_list(amp)
Amplifier **amp;
{
        if (*amp == NULL) return (NULL);
        return ((*amp)->next);
}


Amplifier *prev_amp_list(amp)
Amplifier**amp;
{
        if (*amp == NULL) return (NULL);
        return ((*amp)->prev);
}


Amplifier *add_amp_amp_list(add_amp,net)
Amplifier *add_amp;
Network         **net;
{
        Amplifier *i;
        Amplifier*amp_list;

        amp_list = get_amp_list_network(net);

        if (amp_list == NULL) set_amp_list_network(add_amp,net);
        else
        {
                for (i = amp_list; i->next != NULL; i = next_amp_list(&i));
                add_amp->prev = i;
                add_amp->next = NULL;
                i->next = add_amp;
        }
        return (add_amp);
}


Amplifier *add_and_create_amp_list(net)
Network**net;
{
```

```
 Amplifier *temp;

 temp = create_amp();

 return(add_amp_amp_list (temp, net));
}


void del_amp_amp_list(del_amp, net)
Amplifier *del_amp;
Network         **net;
{
          Amplifier*amp_list;

          amp_list = get_amp_list_network(net);

          if ((del_amp != NULL) && (amp_list != NULL))
          {
                    if (del_amp == amp_list)
                    {
                              set_amp_list_network(del_amp->next,net);
                              if (del_amp->next != NULL) (del_amp->next)->prev = NULL;
                    }
                    else
                    {
 if (del_amp->prev != NULL) (del_amp->prev)->next = del_amp->next;
 if (del_amp->next != NULL) (del_amp->next)->prev = del_amp->prev;
                    }
          }
 free(del_amp->name);
 free(del_amp->location);
 free(del_amp->pilot);
 free(del_amp);
 del_amp = NULL;
}


void destroy_amp_list(net)
Network **net;
/****************************************************************
 * Description: *
 *        This procedure releases memory used fo the amplifier structure. *
 * Error Handling: *
 * None. *
 * Returns: *
 * None. *
 ****************************************************************/
{
          Amplifier *i;
```

```
	for (i=get_amp_list_network(net); i != NULL; i = next_amp_list(&i))
del_amp_amp_list(i,net);
}


Amplifier *find_amp_amp_list(amp_name,net)
char		*amp_name;
Network		**net;
{
	Amplifier*i;

	for (i=get_amp_list_network(net); ((i != NULL) &&
(strcmp(get_name_amp(&i),amp_name) != 0)); i = next_amp_list(&i));

	return (i);
}


void set_all_amp(name,location,type,SMT_num,subs,pilot,termination,page,amp)
char		*name;
char		*location;
char		*type;
int		SMT_num;
int		subs;
char		*pilot;
boolean		termination;
int		page;
Amplifier**amp;
{
	set_name_amp(name,amp);
	set_location_amp(location,amp);
	set_type_amp(type,amp);
	set_SMT_num_amp(SMT_num,amp);
	set_subs_amp(subs,amp);
	set_pilot_amp(pilot,amp);
	set_termination_amp(termination,amp);
	set_page_marker_amp(page,amp);
}


Amplifier
*create_and_set_all_amp(name,location,type,SMT_num,subs,pilot,termination,page,net)
char		*name;
char		*location;
char		*type;
int		SMT_num;
int		subs;
char		*pilot;
boolean		termination;
int		page;
Network		**net;
```

```
{
        Amplifier*temp;

        if (name == ““)
        {
                fprintf(stdout,”ERROR - in create and set_all_amp NAME invalid\n”);
        }
        temp = find_amp_amp_list(name,net);
        if (temp == NULL) temp = add_and_create_amp_list(net);
        set_all_amp(name,location,type,SMT_num,subs,pilot,termination,page,&temp);

        return(temp);
}
```

A.6 Sub Network List Routines (AN_Sub_Network_list.c)

```
#include <strings.h>
#include “AN_defs.h”
#include “AN_Amplifier.h”
#include “AN_Sub_Network.h”


/*****************************************************************
 * *
 * Sub Network list Routines. *
 * *
 *****************************************************************/



Sub_Network_list *create_sub_network_list ()
/*****************************************************************
 * Description: *
 *       This procedure cerates a sub network list object. *
 * Error Handling: *
 * None. *
 * Returns: *
 * None. *
 *****************************************************************/
{
        Sub_Network_list*temp;

        temp = (Sub_Network_list *) malloc (sizeof (Sub_Network_list));

        temp->subnet = NULL;
        temp->next = NULL;
        temp->prev = NULL;

        return (temp);
}
```

```
Sub_Network *get_subnet_sub_network_list (sub_network_list)
Sub_Network_list**sub_network_list;
/****************************************************************
 * Description: *
 *       This procedure gets a pointer to a subnet stored in the Sub_Network *
 *       list. *
 * Error Handling: *
 * None. *
 * Returns: *
 * pointer to the subnetwork. *
 ****************************************************************/
{
        if (*sub_network_list == NULL) return (NULL);
        return((*sub_network_list)->subnet);
}


Sub_Network *set_subnet_sub_network_list (subnet, sub_network_list)
Sub_Network  *subnet;
Sub_Network_list**sub_network_list;
/****************************************************************
 * Description: *
 *       This procedure sets a pointer to a subnet stored in the Sub_Network *
 *       list. *
 * Error Handling: *
 * None. *
 * Returns: *
 * pointer to the subnetwork. *
 ****************************************************************/
{
        (*sub_network_list)->subnet = subnet;
}


Sub_Network_list *next_sub_network_list (sub_network_list)
Sub_Network_list**sub_network_list;
/****************************************************************
 * Description: *
 *       This procedure returns a pointer to the next element in the subnetwork*
 *       list. *
 * Error Handling: *
 * None. *
 * Returns: *
 * pointer to next element in the sub network list. *
 ****************************************************************/
{
        if (*sub_network_list == NULL) return (NULL);
        return((*sub_network_list)->next);
}
```

```
Sub_Network_list *prev_sub_network_list (sub_network_list)
Sub_Network_list**sub_network_list;
/****************************************************************
 * Description: *
 *       This procedure returns a pointer to the prev element in the subnetwork*
 *       list. *
 * Error Handling: *
 * None. *
 * Returns: *
 * pointer to prev element in the sub network list. *
 ****************************************************************/
{
        if (*sub_network_list == NULL) return (NULL);
        return((*sub_network_list)->prev);
}


/*
Sub_Network_list *add_sub_network_list (add_sub_network, sub_network_list)
Sub_Network_list*add_sub_network;
Sub_Network_list**sub_network_list;
*/
/****************************************************************
 * Description: *
 *       This procedure adds a subnetwork to the subnetwork list. *
 *       Immeadiately after the fisrt element in the list (minimium processing time)*
 * Error Handling: *
 * None. *
 * Returns: *
 * pointer to added subnetwork list element. *
 ****************************************************************/
/*
{
        if (*sub_network_list == NULL) *sub_network_list = add_sub_network;

        else
        {
                if  ((*sub_network_list)->next  !=  NULL)  ((*sub_network_list)->next)-
>prev = add_sub_network;
                add_sub_network->prev = *sub_network_list;
                add_sub_network->next = (*sub_network_list)->next;
                (*sub_network_list)->next = add_sub_network;
        }

        return (add_sub_network);
}
*/
Sub_Network_list *add_sub_network_list (add_sub_network, sub_network_list)
Sub_Network_list*add_sub_network;
```

```
Sub_Network_list**sub_network_list;
/**************************************************************
 * Description: *
 *       This procedure adds a subnetwork to the subnetwork list. *
 *       At the end of the existing list (preserves order of elements) *
 * Error Handling: *
 * None. *
 * Returns: *
 * pointer to added subnetwork list element. *
 **************************************************************/
{
         Sub_Network_list*i;
         if (*sub_network_list == NULL) *sub_network_list = add_sub_network;

         else
         {
                  for (i = *sub_network_list; next_sub_network_list(&i) != NULL; i =
next_sub_network_list(&i));
                  i->next = add_sub_network;
                  add_sub_network->prev = i;
         }

         return (add_sub_network);
}


Sub_Network_list *add_and_create_sub_network_list (sub_network_list)
Sub_Network_list**sub_network_list;
/**************************************************************
 * Description: *
 *       This procedure creates a new subnetwork and adds it to the subnetwork *
 *       list. *
 * Error Handling: *
 * None. *
 * Returns: *
 * pointer to added subnetwork list element. *
 **************************************************************/
{
         Sub_Network_list*temp;

         temp = create_sub_network_list();

         return(add_sub_network_list (temp, sub_network_list));
}


void del_sub_network_list (del_sub_network, sub_network_list)
Sub_Network_list*del_sub_network;
Sub_Network_list**sub_network_list;
/**************************************************************
 * Description: *
```

```
 *       This procedure deletes a subnetwork from the subnetwork list. *
 * Error Handling: *
 * None. *
 * Returns: *
 * None. *
 ***************************************************************/
{
        if ((*sub_network_list != NULL) && (del_sub_network != NULL))
        {
                if (*sub_network_list == del_sub_network)
                {
                        *sub_network_list = del_sub_network->next;
                        if  (del_sub_network->next  !=  NULL)  (del_sub_network->next)-
>prev = NULL;
                }
                else
                {
                        if  (del_sub_network->prev  !=  NULL)  (del_sub_network->prev)-
>next = del_sub_network->next;

                        if  (del_sub_network->next  !=  NULL)  (del_sub_network->next)-
>prev = del_sub_network->prev;
                }
        }

        free (del_sub_network);
        del_sub_network = NULL;
}


void destroy_sub_network_list (sub_network_list)
Sub_Network_list**sub_network_list;
/***************************************************************
 * Description: *
 *       This procedure destroys a sub network list. *
 * Error Handling: *
 * None. *
 * Returns: *
 * None. *
 ***************************************************************/
{
        Sub_Network_list*i;

        if (*sub_network_list != NULL)
        {
                for (i = *sub_network_list; i != NULL; i=i->next)
                {
                        del_sub_network_list(i,sub_network_list);
                }
        }
}
```

```
Sub_Network_list *find_subnet_sub_network_list(subnet,sub_network_list)
Sub_Network *subnet;
Sub_Network_list **sub_network_list;
/*************************************************************
 * Description: *
 *       This procedure finds the sub_network_list with the entry subnet. *
 * Error Handling: *
 * None. *
 * Returns: *
 * Pointer to the sub_network_list. *
 *************************************************************/
{
        Sub_Network_list *i;

        for (i=*sub_network_list; ((i!=NULL) && (get_subnet_sub_network_list(&i) !=
subnet)); i=next_sub_network_list(&i));

        if (i == NULL)
         for (i=*sub_network_list; ((i!=NULL) && (get_subnet_sub_network_list(&i) !=
subnet)); i=prev_sub_network_list(&i));

        return(i);
}


Sub_Network_list *create_and_set_all_sub_network_list(subnet,subnet_list)
Sub_Network  *subnet;
Sub_Network_list**subnet_list;
{

        Sub_Network_list*temp;

        temp = find_subnet_sub_network_list(subnet,subnet_list);
        if (temp == NULL)
        {
                temp = add_and_create_sub_network_list(subnet_list);
                set_subnet_sub_network_list(subnet,&temp);
        }

        return (temp);
}


Sub_Network*find_name_sub_network_list(name,subnet_list)
char                                    *name;
Sub_Network_list**subnet_list;
{
        char                                    *tmp;
        Sub_Network             *subnet;
        Amplifier               *amp;
```

```
	Sub_Network_list*i;

	i		= *subnet_list;
	subnet = get_subnet_sub_network_list(&i);
	amp = get_amp_subnet(&subnet);
	tmp = get_name_amp(&amp);

	while ((next_sub_network_list(&i) != NULL) && (strcmp(tmp,name) != 0))
	{
		i			= next_sub_network_list(&i);
		subnet = get_subnet_sub_network_list(&i);
		amp = get_amp_subnet(&subnet);
		tmp = get_name_amp(&amp);
	}

	if ((tmp != NULL) && (strcmp(tmp,name) == 0)) return(subnet);
	return (NULL);
}
```

Appendix B:

Cluster Server C Routines

The following are the listings of the C source code developed to implement the cluster server.

B.1 Cluster Server Routines (Cluster.c)

```
#include <stdio.h>
#include "cluster_defs.h"
#include "amp_name_list.h"
#include "../topology_server/net_topology/AN_topology.h"
#include "../topology_server/topology_struct.h"
#include "../data_server/lg_data/lgf_access_defs.h"
#include "../data_server/lg_data/log_data_defs.h"
#include "../data_server/lg_data/log_data.h"
#include "../data_server/lg_data/lgf_access.h"
#include "../fault_server/lgf_control/lgf_control.h"
#include "../fault_server/lgf_control/lgf_control_defs.h"
#include "/home/awatkins/play/ui/nxp_control/control_nxp.h"
#include "/home/awatkins/play/nxp_load/load_nxp.h"
#include "/home/awatkins/play/Rog_stats/retreive/data_record.h"

#define DO_INITIAL_CLUSTER-1

static Network*net;

void    init_all()
{
        net = read_network("/net/eros/eros/m/rogers/newdata/rogers.net");
        create_amp_name_list();

        NXCinit_nexpert();
        NXCload_unload_kb(PROTOTYPE_KB_NAME, LOAD_KB);
        CPGinstall_handler();
        CSinstall_handlers();
        TSinstall_handler();
        MMVinstall_handler();
}


void get_fault_info_from_user (do_another, yr, mo, day, amp, fault_time, window)
int *do_another, *yr, *mo, *day;
char *amp;
```

```
double *fault_time;
int *window;
/*****************************************************
 * Description: *
 *       This routine prompts the user to enter info about which fault will be loaded *
 *       into Nexpert. After the initial cluster, the user is first prompted whether *
 *       or not another cluster is to be loaded.  *
 * Error Handling: *
 * Returns: *
 *****************************************************/
{
 char continue_str[40];

 if (*do_another != DO_INITIAL_CLUSTER)
 {
 fprintf (stdout, "Do you want to load another cluster?\n");
 fscanf (stdin, "%s", continue_str);
 if ((continue_str[0] == 'y') || (continue_str[0] == 'Y'))
         *do_another = TRUE;
 else
         *do_another = FALSE;
 }
 else
 *do_another = TRUE;
 if (*do_another)
 {
 fprintf (stdout, "Enter a new fault. Separate entries with spaces.\n\n");
 fprintf (stdout, "Enter year, month and day of the fault's log file: ");
 fscanf (stdin, "%d%d%d", yr, mo, day);
 fprintf (stdout, "Enter the failed amp and its fault time: ");
 fscanf (stdin, "%s%lf", amp, fault_time);
 fprintf (stdout, "Enter the desired time window: ");
 fscanf (stdin, "%d", window);
 fprintf (stdout, "\n");
 }
}



int      days_in_mon(mon)
int      mon;
{
 int days_in_mon;

 days_in_mon = 31;

 if (mon == 2) days_in_mon = 28;
 if (mon == 9) days_in_mon = 30;
 if (mon == 4) days_in_mon = 30;
 if (mon == 6) days_in_mon = 30;
 if (mon == 11) days_in_mon = 30;
```

```
        return(days_in_mon);
}

void    get_start_end_times(fault_time,window,start_time,end_time)
double fault_time;
int             window;
double *start_time;
double *end_time;
{
 int start_month, start_day, start_hour, start_min;
 int end_month, end_day, end_hour, end_min;
 int start_minutes = 0;
 int start_hours,start_days;
 int end_minutes,end_hours,end_days;
 char start_time_str[20];
        int     i;

 sprintf(start_time_str,"%8.0f",fault_time);
 if (start_time_str[0] == ' ') start_time_str[0] = '0';

  sscanf (start_time_str, "%2d%2d%2d%2d", &start_month, &start_day, &start_hour,
&start_min);

        for (i=1; i<start_month; i++)
                start_minutes = start_minutes + days_in_mon(i)*24*60;
 start_minutes = start_minutes + ((start_day-1) * 24 * 60) + (start_hour * 60) + start_min;

        end_minutes = start_minutes + window;
        end_min = end_minutes%60;
        end_hours = end_minutes/60;
        end_hour = end_hours%24;
        end_days = end_hours/24;
        end_days++;


        for (i=1;end_days > days_in_mon(i); i++)
                end_days = end_days - days_in_mon(i);

        if (i > 12) i = i - 12;

        *end_time = (((i*100)+end_days)*100+end_hour)*100+end_min;

        if ((start_minutes - window )< 0)
         start_minutes = (365*24*60) - window + start_minutes;
        else
         start_minutes = start_minutes - window;
        start_min = start_minutes%60;
        start_hours = start_minutes/60;
        start_hour = start_hours%24;
        start_days = start_hours/24;
```

```
 start_days++;
        for (i=1;start_days > days_in_mon(i); i++)
                start_days = start_days - days_in_mon(i);

        if (i > 12) i = i - 12;
        *start_time = (((i*100)+start_days)*100+start_hour)*100+start_min;
}


int     check_log_data_for_problem(tmp)
log_data*tmp;
{
        int     max;
        int     i;

        max = get_num_log_values_added(tmp);

        for (i=0; i<max; i++)
        {
                if
(strcmp(get_ith_str_from_log_data(tmp,FAULT_TYPE_STR_FIELD,i),ALARM_FAUL
T_MESSAGE)) return(1);
                if
(strcmp(get_ith_str_from_log_data(tmp,FAULT_TYPE_STR_FIELD,i),FAIL_FAULT_
MESSAGE)) return(1);
                if
(strcmp(get_ith_str_from_log_data(tmp,FAULT_TYPE_STR_FIELD,i),WARN_FAULT
_MESSAGE)) return(1);
        }

        return(0);
}



int     search_string_array(item,max,array)
char    *item;
int     max;
char    *array[];
{
        int     i;

        if (array == NULL) return(0);
        for (i=0; i<max; i++)
                if (strcmp(item,array[i]) == 0) return(1);
        return(0);
}


int     make_cluster(fault_amp,start_time,end_time)
```

```
char		*fault_amp;
double start_time;
double end_time;
{
	log_data	*tmp_data = NULL;
	log_data	*log_pak = NULL;
	topol_structtopol_pak;
	data_struct	data_pak;
	char			*possible_fault[MAX_CONNECTED_AMPS];
	Amplifier*amp,*parent_amp;
	Sub_Network*curr_subnet;
	Sub_Network*parent_subnet;
	Power_Grid*power_grid;
	Sub_Network_list*children_list,*power_grid_list,*net_list;
	int			power_grid_id;
	int			i,l;
	Sub_Network_list*j,*k;
	Sub_Network*subnet;
	char	*name;
	char	start_time_str[20];
	char	end_time_str[20];

	net_list		= get_subnet_list_network(&net);
	curr_subnet	= find_name_sub_network_list(fault_amp,&net_list);
	if (curr_subnet == NULL)
	{
		fprintf(stdout,"ERROR - invalid amplifier %s specified MAKE_CLUSTER
routine\n",fault_amp);
		return (0);
	}
	fprintf(stdout,"%s is in cluster \n",fault_amp);

	parent_subnet = get_prev_subnet(&curr_subnet);
	power_grid = get_power_grid_subnet(&curr_subnet);
	children_list = get_branches_subnet(&curr_subnet);
	power_grid_id = get_power_grid_id_power_grid(&power_grid);
	power_grid_list = get_subnet_list_power_grid(&power_grid);
	parent_amp = get_amp_subnet(&parent_subnet);

	insert_amp_name(fault_amp);

	log_pak = get_log_file_data(fault_amp,start_time,end_time);

	amp = get_amp_subnet(&curr_subnet);
	topol_pak.amp_name = get_name_amp(&amp);
 topol_pak.location = get_location_amp(&amp);
 topol_pak.type = get_type_amp(&amp);
 topol_pak.SMT_num = get_SMT_num_amp(&amp);
 topol_pak.subs = get_subs_amp(&amp);
 topol_pak.pilot = get_pilot_amp(&amp);
	/* change line to check for both TERMINAL and NEXT to terminal
```

```
                amp conditions, set topol_pak.termination to TRUE in either
                case (boolean topol_pak.termination) */
topol_pak.termination = get_termination_amp(&amp);
topol_pak.parent = get_name_amp(&parent_amp);
topol_pak.power_grid = power_grid_id;

sprintf(start_time_str,"%8.0f",start_time);
if (start_time_str[0] == ' ') start_time_str[0] = '0';
sprintf(end_time_str,"%8.0f",end_time);
if (end_time_str[0] == ' ') end_time_str[0] = '0';

        AWQprime_data_pak(&data_pak,fault_amp,start_time_str,end_time_str);
        QARretreive_data(&data_pak);
AWQprint_data_pak (&data_pak);
        NXLcreate_amp_load_values(&data_pak,log_pak,&topol_pak);
        AWQfree_data_pak_arrays (&data_pak);

        del_log_data(log_pak);

        i = 0;
        amp = get_amp_subnet(&parent_subnet);
        possible_fault[i] = get_name_amp(&amp);
        if (possible_fault[i] == NULL) possible_fault[i] = "";
        i++;

        for (j=power_grid_list; j != NULL; j = next_sub_network_list(&j))
        {
                subnet = get_subnet_sub_network_list(&j);
                amp = get_amp_subnet(&subnet);
                name = get_name_amp(&amp);
                if (!(search_string_array(name,i,possible_fault)))
                {
                        possible_fault[i] = name;
                        i++;
                }
        }

        for (k=children_list; k != NULL; k = next_sub_network_list(&k))
        {
                subnet = get_subnet_sub_network_list(&k);
                amp = get_amp_subnet(&subnet);
                name = get_name_amp(&amp);
                if (!(search_string_array(name,i,possible_fault)))
                {
                        possible_fault[i] = name;
                        i++;
                }
        }

        for (l=0; l <i; l++)
        {
```

```
                fprintf(stdout,"\tChecking %s \n",possible_fault[l]);

                if (search_amp_name_list(possible_fault[l]) == 0)
                {
                        tmp_data                                        =
get_log_file_data(possible_fault[l],start_time,end_time);
                        if (check_log_data_for_problem(tmp_data))
                        {
                                make_cluster(possible_fault[l],start_time,end_time);
                        }
                        del_log_data(tmp_data);
                }
        }
        return(1);
}


void fatal_error (str1, str2)
char *str1, *str2;
/*******************************************************
 *******************************************************/

{
 fprintf (stderr, "FATAL ERROR: %s%s\n", str1, str2);
 exit (-1);
}



void warning (str1, str2)
char *str1, *str2;
/*******************************************************
 *******************************************************/
{
 fprintf (stderr, "WARNING: %s%s\n", str1, str2);
}


main(argc,argv)
int     argc;
char    *argv[];
/*******************************************************
 *******************************************************/
{
 FILE_STRUCT*open_file = NULL;
 char   tmp_string[4];

 int year;
 int mon;
 int day;
```

```
 char amp_name[8], user_entered_amp[10];
 char fault_type[20];
 char fault_message[20];
 char fault_value[20];
 char fault_nominal[20];
 double fault_time, user_entered_fail_time;

 double start_time,end_time;
 int           window;
 int           start, do_another_cluster;

 int           i = 1;

 init_all();
/*
 year = atoi(argv[1]);
 mon = atoi(argv[2]);
 day = atoi(argv[3]);

 window = atoi(argv[4]);
 year = 91;
 mon = 12;
 day = 16;
 window = 15;
*/

 do_another_cluster = DO_INITIAL_CLUSTER;
 while (do_another_cluster)
 {
    get_fault_info_from_user (&do_another_cluster, &year, &mon, &day,
user_entered_amp,
                    &user_entered_fail_time, &window);
 if (do_another_cluster)
 {
 start = 0;
 while ((get_next_fault(&year,&mon,&day,amp_name,fault_type,fault_message,
                    fault_value,fault_nominal,&fault_time,&open_file,tmp_string) ==
0)
 && (start == 0))
 {
/*
 fprintf(stdout,"%s %s %s %s %s %f\n",amp_name,fault_type,fault_message,
                              fault_value,fault_nominal,fault_time);
*/
   if ((strcmp(amp_name,user_entered_amp) == 0) && (fault_time ==
user_entered_fail_time))
 start = 1;
 if (start)
 {
 fprintf(stdout,"*** CLUSTER %d ****\n",i);
 i++;
```

```
 get_start_end_times(fault_time,window,&start_time,&end_time);
/*
 fprintf(stdout,"Start time = %f End Time = %f\n",start_time,end_time);
*/
 NXLset_state_variables(amp_name,fault_type,fault_message,fault_value,
                                        fault_nominal,fault_time);
 create_amp_name_list();
 make_cluster(amp_name,start_time,end_time);
 delete_amp_name_list();
 NXCstart_graphics();
 NXCunload_temporary_kb();
            }
 }
 }
 }
}
```

B.2 Amplifier List C Routines (amp_name_list.c)

```
#include <stdio.h>
#include <strings.h>
#include "../fault_server/lgf_control/lgf_control_defs.h"
#include "list_defs.h"

static  int     num;
static  char    *amp_name_list = NULL;

void    create_amp_name_list()
{
        num = 0;
        amp_name_list = (char *) calloc (NUM_AMPS,sizeof(char)*AMP_NAME_LEN);
}


void    insert_amp_name(amp)
char    *amp;
{
        strcpy(amp_name_list+sizeof(char)*AMP_NAME_LEN*num,amp);
        num++;
}


void    delete_amp_name_list()
{
        free(amp_name_list);
        amp_name_list = NULL;
}
```

```
int     search_amp_name_list(amp)
char    *amp;
{
        int     i;

        for (i=0; i < num; i++)
        {
                if (strcmp(amp_name_list+sizeof(char)*AMP_NAME_LEN*i,amp) == 0)
                        return(1);
        }
        return(0);
}
```

Appendix C:

Diagnostic Tool Knowledge Bases

The following are the listing of the knowledge bases used by the diagnostic tool.

C.1 Diagnostic Knowledge Base (diag_2.kb)

```
(@VERSION=020)
(@PROPERTY=max_affected@TYPE=Integer;)
(@PROPERTY=max_dist@TYPE=Integer;)
(@PROPERTY=min_affected@TYPE=Integer;)
(@PROPERTY=min_dist@TYPE=Integer;)
(@PROPERTY=name@TYPE=String;)
(@PROPERTY=num_paths@TYPE=Integer;)



(@OBJECT=All_Amps_in_Power_Grid_Effected
	(@PROPERTIES=
		Value	@TYPE=Boolean;
	)
)

(@OBJECT=all_Amps_on_Power_Grid_Effected
	(@PROPERTIES=
		Value	@TYPE=Boolean;
	)
)

(@OBJECT=Amp_Communicating
	(@PROPERTIES=
		Value	@TYPE=Boolean;
	)
)

(@OBJECT=Amp_component_Failed
	(@PROPERTIES=
		Value	@TYPE=Boolean;
	)
)

(@OBJECT=Amp_component_not_working_properly
	(@PROPERTIES=
		Value	@TYPE=Boolean;
	)
```

```
)

(@OBJECT=Amp_is_communicating
	(@PROPERTIES=
		Value	@TYPE=Boolean;
	)
)

(@OBJECT=Amp_NOT_Communicating
	(@PROPERTIES=
		Value	@TYPE=Boolean;
	)
)

(@OBJECT=Amp_NOT_Communictating
	(@PROPERTIES=
		Value	@TYPE=Boolean;
	)
)

(@OBJECT=Amp_out_of_alignment
	(@PROPERTIES=
		Value	@TYPE=Boolean;
	)
)

(@OBJECT=amp_to_term
	(@PROPERTIES=
		max_dist
		min_dist
		num_paths
	)
)

(@OBJECT=Amplifier_Failure
	(@PROPERTIES=
		Value	@TYPE=Boolean;
	)
)

(@OBJECT=Are_all_amps_on_power_grid_effected
	(@PROPERTIES=
		Value	@TYPE=Boolean;
	)
)

(@OBJECT=Are_there_Downstream_Effects
	(@PROPERTIES=
		Value	@TYPE=Boolean;
	)
)
```

```
(@OBJECT=Are_there_more_than_4_downstream_amps_effected
	(@PROPERTIES=
		Value	@TYPE=Boolean;
	)
)

(@OBJECT=Are_we_at_the_terminal_amp
	(@PROPERTIES=
		Value	@TYPE=Boolean;
	)
)

(@OBJECT=Are_we_within_4_amps_of_the_Terminal
	(@PROPERTIES=
		Value	@TYPE=Boolean;
	)
)

(@OBJECT=at_Terminal_Amp
	(@PROPERTIES=
		Value	@TYPE=Boolean;
	)
)

(@OBJECT=Bad_Connection
	(@PROPERTIES=
		Value	@TYPE=Boolean;
	)
)

(@OBJECT=check_
	(@PROPERTIES=
		Value	@TYPE=Boolean;
	)
)

(@OBJECT=check_All_Amps_on_Power_Grid_Effected
	(@PROPERTIES=
		Value	@TYPE=Boolean;
	)
)

(@OBJECT=check_Amp_Communicating
	(@PROPERTIES=
		Value	@TYPE=Boolean;
	)
)

(@OBJECT=check_Are_all_Amps_on_Power_Grid_Effected
	(@PROPERTIES=
```

```
                Value   @TYPE=Boolean;
        )
)

(@OBJECT=check_Downstream_Amp_Communicating
        (@PROPERTIES=
                Value   @TYPE=Boolean;
        )
)

(@OBJECT=check_Downstream_Amp_Communication
        (@PROPERTIES=
                Value   @TYPE=Boolean;
        )
)

(@OBJECT=check_Downstream_Amp_Communictating
        (@PROPERTIES=
                Value   @TYPE=Boolean;
        )
)

(@OBJECT=check_for_Downstream_Effects
        (@PROPERTIES=
                Value   @TYPE=Boolean;
        )
)

(@OBJECT=check_for_Terminal_Amp
        (@PROPERTIES=
                Value   @TYPE=Boolean;
        )
)

(@OBJECT=check_for_Warning_Ok_Cycle
        (@PROPERTIES=
                Value   @TYPE=Boolean;
        )
)

(@OBJECT=check_Is_Any_Downstream_Amp_Communicating
        (@PROPERTIES=
                Value   @TYPE=Boolean;
        )
)

(@OBJECT=check_More_than_4_downstream_Amps_Effected
        (@PROPERTIES=
                Value   @TYPE=Boolean;
        )
)
```

```
(@OBJECT=check_Upstream_Amp_Low_Pilot
        (@PROPERTIES=
                Value   @TYPE=Boolean;
        )
)

(@OBJECT=check_Warning_Ok_Cycle
        (@PROPERTIES=
                Value   @TYPE=Boolean;
        )
)

(@OBJECT=check_Within_4_Amps_of_Terminal
        (@PROPERTIES=
                Value   @TYPE=Boolean;
        )
)

(@OBJECT=chek_Upstream_Amp_Low_Pilot
        (@PROPERTIES=
                Value   @TYPE=Boolean;
        )
)

(@OBJECT=Communication_Problem_between_head_Amp_and_low_Pilots_parent
        (@PROPERTIES=
                Value   @TYPE=Boolean;
        )
)

(@OBJECT=Communication_problem_between_head_amp_and_parent
        (@PROPERTIES=
                Value   @TYPE=Boolean;
        )
)

(@OBJECT=Dead_Amplifier
        (@PROPERTIES=
                Value   @TYPE=Boolean;
        )
)

(@OBJECT=Diagnose_Fault
        (@PROPERTIES=
                Value   @TYPE=Boolean;
        )
)

(@OBJECT=Display_ACF_message
        (@PROPERTIES=
```

```
                Value  @TYPE=Boolean;
        )
)

(@OBJECT=display_ACNWP_message
        (@PROPERTIES=
                Value  @TYPE=Boolean;
        )
)

(@OBJECT=display_AOOA_message
        (@PROPERTIES=
                Value  @TYPE=Boolean;
        )
)

(@OBJECT=display_BC_message
        (@PROPERTIES=
                Value  @TYPE=Boolean;
        )
)

(@OBJECT=display_CPBHAALPP_message
        (@PROPERTIES=
                Value  @TYPE=Boolean;
        )
)

(@OBJECT=display_CPBHAAP_message
        (@PROPERTIES=
                Value  @TYPE=Boolean;
        )
)

(@OBJECT=display_DA_message
        (@PROPERTIES=
                Value  @TYPE=Boolean;
        )
)

(@OBJECT=display_FIA_message
        (@PROPERTIES=
                Value  @TYPE=Boolean;
        )
)

(@OBJECT=display_FIOFHA_message
        (@PROPERTIES=
                Value  @TYPE=Boolean;
        )
)
```

```
(@OBJECT=display_HO_message
	(@PROPERTIES=
		Value  @TYPE=Boolean;
	)
)

(@OBJECT=display_LB_message
	(@PROPERTIES=
		Value  @TYPE=Boolean;
	)
)

(@OBJECT=display_NNTBR_message
	(@PROPERTIES=
		Value  @TYPE=Boolean;
	)
)

(@OBJECT=display_PSF_message
	(@PROPERTIES=
		Value  @TYPE=Boolean;
	)
)

(@OBJECT=display_UP_message
	(@PROPERTIES=
		Value  @TYPE=Boolean;
	)
)

(@OBJECT=Downstream_Amp_Communicating
	(@PROPERTIES=
		Value  @TYPE=Boolean;
	)
)

(@OBJECT=Downstream_Amp_NOT_Communicating
	(@PROPERTIES=
		Value  @TYPE=Boolean;
	)
)

(@OBJECT=Downstream_Effects
	(@PROPERTIES=
		Value  @TYPE=Boolean;
	)
)

(@OBJECT=downstream_paths
	(@PROPERTIES=
```

```
        max_affected
        min_affected
        num_paths
    )
)

(@OBJECT=Failure_in_Amp
    (@PROPERTIES=
        Value   @TYPE=Boolean;
    )
)

(@OBJECT=Failure_in_Amplifier
    (@PROPERTIES=
        Value   @TYPE=Boolean;
    )
)

(@OBJECT=Failure_in_output_of_Head_Amp
    (@PROPERTIES=
        Value   @TYPE=Boolean;
    )
)

(@OBJECT=goto_Head_Amp
    (@PROPERTIES=
        Value   @TYPE=Boolean;
    )
)

(@OBJECT=head_amp
    (@PROPERTIES=
        name
    )
)

(@OBJECT=Hydro_Outage
    (@PROPERTIES=
        Value   @TYPE=Boolean;
    )
)

(@OBJECT=Is_Amp_Communictating
    (@PROPERTIES=
        Value   @TYPE=Boolean;
    )
)

(@OBJECT=Is_any_downstream_amp_communicating
    (@PROPERTIES=
        Value   @TYPE=Boolean;
```

```
	)
)

(@OBJECT=Is_the_Upstream_Amp_a_Low_Pilot
	(@PROPERTIES=
		Value	@TYPE=Boolean;
	)
)

(@OBJECT=Is_there_a_Warning_OK_Cycle
	(@PROPERTIES=
		Value	@TYPE=Boolean;
	)
)

(@OBJECT=Line_Break
	(@PROPERTIES=
		Value	@TYPE=Boolean;
	)
)

(@OBJECT=More_than_4_Downstream_Amps_Effected
	(@PROPERTIES=
		Value	@TYPE=Boolean;
	)
)

(@OBJECT=Nominals_need_to_be_reset
	(@PROPERTIES=
		Value	@TYPE=Boolean;
	)
)

(@OBJECT=Parent
	(@PROPERTIES=
		pilot
		Value	@TYPE=String;
	)
)

(@OBJECT=Power_Supply_Failure
	(@PROPERTIES=
		Value	@TYPE=Boolean;
	)
)

(@OBJECT=Set_Amp_Communicating
	(@PROPERTIES=
		Value	@TYPE=Boolean;
	)
)
```

```
(@OBJECT=Set_Downstream_Amp_Communicating
	(@PROPERTIES=
		Value	@TYPE=Boolean;
	)
)

(@OBJECT=Set_Upstream_Amp_Low_Pilot
	(@PROPERTIES=
		Value	@TYPE=Boolean;
	)
)

(@OBJECT=terminal_amp
	(@PROPERTIES=
		Value	@TYPE=Boolean;
	)
)

(@OBJECT=Unknown_Problem
	(@PROPERTIES=
		Value	@TYPE=Boolean;
	)
)

(@OBJECT=Upstream_Amp_Low_Pilot
	(@PROPERTIES=
		Value	@TYPE=Boolean;
	)
)

(@OBJECT=Upstream_Low_Pilot
	(@PROPERTIES=
		Value	@TYPE=Boolean;
	)
)

(@OBJECT=Warning_Ok_Cycle
	(@PROPERTIES=
		Value	@TYPE=Boolean;
	)
)

(@OBJECT=Within_4_Amps_of_Terminal
	(@PROPERTIES=
		Value	@TYPE=Boolean;
	)
)

(@RULE=R1
	(@LHS=
```

```
                (Yes    (check_All_Amps_on_Power_Grid_Effected))
                (Equal  (<|topology_data|>.power_grid)
(‘topol_’\head_amp.name\.power_grid))
                (Execute(“all_amps_on_PG_loaded”)
(@ATOMID=<|topology_data|>.power_grid;))
        )
        (@HYPO=all_Amps_on_Power_Grid_Effected)
)

(@RULE=R2
        (@LHS=
                (No     (Amp_NOT_Communicating))
        )
        (@HYPO=Amp_Communicating)
)

(@RULE=R4
        (@LHS=
                (No     (at_Terminal_Amp))
                (No     (Downstream_Effects))
                (No     (all_Amps_on_Power_Grid_Effected))
                (No     (Amp_Communicating))
        )
        (@HYPO=Amp_component_Failed)
)

(@RULE=R3
        (@LHS=
                (No     (at_Terminal_Amp))
                (No     (Downstream_Effects))
                (No     (all_Amps_on_Power_Grid_Effected))
                (Yes    (Amp_Communicating))
                (No     (Warning_Ok_Cycle))
        )
        (@HYPO=Amp_component_Failed)
)

(@RULE=R6
        (@LHS=
                (No     (Downstream_Effects))
                (No     (all_Amps_on_Power_Grid_Effected))
                (Yes    (Amp_Communicating))
                (Yes    (Warning_Ok_Cycle))
        )
        (@HYPO=Amp_component_not_working_properly)
)

(@RULE=R5
        (@LHS=
                (Yes    (at_Terminal_Amp))
                (No     (all_Amps_on_Power_Grid_Effected))
```

```
		(Yes	(Amp_Communicating))
		(Yes	(Warning_Ok_Cycle))
	)
	(@HYPO=Amp_component_not_working_properly)
)

(@RULE=R7
	(@LHS=
		(Yes	(check_Amp_Communicating))
		(Execute("TestMultiValue")
(@ATOMID='log_'\head_amp.name\.fault_message;\
@STRING="@ANY=ALL,@TEST=NO REPLY,@COMP=STRING";\
))
	)
	(@HYPO=Amp_NOT_Communicating)
)

(@RULE=R9
	(@LHS=
		(No	(Downstream_Effects))
		(No	(all_Amps_on_Power_Grid_Effected))
		(Yes	(Amp_Communicating))
		(Yes	(Warning_Ok_Cycle))
	)
	(@HYPO=Amp_out_of_alignment)
)

(@RULE=R8
	(@LHS=
		(Yes	(at_Terminal_Amp))
		(No	(all_Amps_on_Power_Grid_Effected))
		(Yes	(Amp_Communicating))
		(Yes	(Warning_Ok_Cycle))
	)
	(@HYPO=Amp_out_of_alignment)
)

(@RULE=R10
	(@LHS=
		(Yes	(check_for_Terminal_Amp))
		(=	('topol_'\head_amp.name\.termination)(-1))
	)
	(@HYPO=at_Terminal_Amp)
)

(@RULE=R12
	(@LHS=
		(Yes	(Downstream_Effects))
		(Yes	(Within_4_Amps_of_Terminal))
		(No	(all_Amps_on_Power_Grid_Effected))
		(Yes	(Downstream_Amp_Communicating))
```

```
        )
        (@HYPO=Bad_Connection)
)

(@RULE=R11
        (@LHS=
                (Yes    (Downstream_Effects))
                (Yes    (Within_4_Amps_of_Terminal))
                (Yes    (all_Amps_on_Power_Grid_Effected))
                (Yes    (Downstream_Amp_Communicating))
        )
        (@HYPO=Bad_Connection)
)

(@RULE=R16
        (@LHS=
                (Yes    (Downstream_Effects))
                (No     (Within_4_Amps_of_Terminal))
                (Yes    (More_than_4_Downstream_Amps_Effected))
        )
        (@HYPO=check_All_Amps_on_Power_Grid_Effected)
)

(@RULE=R15
        (@LHS=
                (Yes    (Downstream_Effects))
                (Yes    (Within_4_Amps_of_Terminal))
        )
        (@HYPO=check_All_Amps_on_Power_Grid_Effected)
)

(@RULE=R14
        (@LHS=
                (Yes    (at_Terminal_Amp))
        )
        (@HYPO=check_All_Amps_on_Power_Grid_Effected)
)

(@RULE=R13
        (@LHS=
                (No     (at_Terminal_Amp))
                (No     (Downstream_Effects))
        )
        (@HYPO=check_All_Amps_on_Power_Grid_Effected)
)

(@RULE=R19
        (@LHS=
                (Yes    (Downstream_Effects))
                (No     (Within_4_Amps_of_Terminal))
                (No     (More_than_4_Downstream_Amps_Effected))
```

```
        )
        (@HYPO=check_Amp_Communicating)
        (@RHS=
                (Let    (Power_Supply_Failure)(FALSE))
                (Let    (Line_Break)(FALSE))
        )
)

(@RULE=R18
        (@LHS=
                (Yes    (at_Terminal_Amp))
                (No     (all_Amps_on_Power_Grid_Effected))
        )
        (@HYPO=check_Amp_Communicating)
)

(@RULE=R17
        (@LHS=
                (No     (at_Terminal_Amp))
                (No     (Downstream_Effects))
                (No     (all_Amps_on_Power_Grid_Effected))
        )
        (@HYPO=check_Amp_Communicating)
)

(@RULE=R22
        (@LHS=
                (Yes    (Downstream_Effects))
                (No     (Within_4_Amps_of_Terminal))
                (Yes    (More_than_4_Downstream_Amps_Effected))
                (Yes    (all_Amps_on_Power_Grid_Effected))
        )
        (@HYPO=check_Downstream_Amp_Communicating)
)

(@RULE=R21
        (@LHS=
                (Yes    (Downstream_Effects))
                (Yes    (Within_4_Amps_of_Terminal))
        )
        (@HYPO=check_Downstream_Amp_Communicating)
)

(@RULE=R20
        (@LHS=
                (Yes    (Downstream_Effects))
                (No     (Within_4_Amps_of_Terminal))
                (Yes    (More_than_4_Downstream_Amps_Effected))
                (Yes    (all_Amps_on_Power_Grid_Effected))
        )
        (@HYPO=check_Downstream_Amp_Communicating)
```

```
)

(@RULE=R23
        (@LHS=
                (No     (at_Terminal_Amp))
        )
        (@HYPO=check_for_Downstream_Effects)
)

(@RULE=R24
        (@LHS=
                (Yes    (goto_Head_Amp))
        )
        (@HYPO=check_for_Terminal_Amp)
)

(@RULE=R26
        (@LHS=
                (No     (at_Terminal_Amp))
                (No     (Downstream_Effects))
                (No     (all_Amps_on_Power_Grid_Effected))
                (Yes    (Amp_Communicating))
        )
        (@HYPO=check_for_Warning_Ok_Cycle)
)

(@RULE=R25
        (@LHS=
                (Yes    (at_Terminal_Amp))
                (No     (all_Amps_on_Power_Grid_Effected))
                (Yes    (Amp_Communicating))
        )
        (@HYPO=check_for_Warning_Ok_Cycle)
)

(@RULE=R27
        (@LHS=
                (Yes    (Downstream_Effects))
                (No     (Within_4_Amps_of_Terminal))
        )
        (@HYPO=check_More_than_4_downstream_Amps_Effected)
)

(@RULE=R28
        (@LHS=
                (Yes    (Downstream_Effects))
                (No     (Within_4_Amps_of_Terminal))
                (No     (More_than_4_Downstream_Amps_Effected))
                (Yes    (Amp_Communicating))
        )
        (@HYPO=check_Upstream_Amp_Low_Pilot)
```

```
)

(@RULE=R29
        (@LHS=
                (Yes    (Downstream_Effects))
        )
        (@HYPO=check_Within_4_Amps_of_Terminal)
)

(@RULE=R30
        (@LHS=
                (Yes    (Downstream_Effects))
                (No     (Within_4_Amps_of_Terminal))
                (No     (More_than_4_Downstream_Amps_Effected))
                (Yes    (Amp_Communicating))
                (Yes    (Upstream_Amp_Low_Pilot))
        )
        (@HYPO=
Communication_Problem_between_head_Amp_and_low_Pilots_parent)
)

(@RULE=R31
        (@LHS=
                (Yes    (Downstream_Effects))
                (No     (Within_4_Amps_of_Terminal))
                (No     (More_than_4_Downstream_Amps_Effected))
                (Yes    (Amp_Communicating))
                (No     (Upstream_Amp_Low_Pilot))
        )
        (@HYPO=Communication_problem_between_head_amp_and_parent)
)

(@RULE=R32
        (@LHS=
                (Yes    (at_Terminal_Amp))
                (No     (all_Amps_on_Power_Grid_Effected))
                (No     (Amp_Communicating))
        )
        (@HYPO=Dead_Amplifier)
)

(@RULE=R33
        (@LHS=
                (Yes    (Amp_component_Failed))
        )
        (@HYPO=Display_ACF_message)
        (@RHS=
                (Show  (“Amp_component_Failed.txt”)
(@KEEP=FALSE;@WAIT=TRUE;))
        )
)
```

```
(@RULE=R34
        (@LHS=
                (Yes    (Amp_component_not_working_properly))
        )
        (@HYPO=display_ACNWP_message)
        (@RHS=
                (Show  ("Amp_component_not_working_properly.txt")
(@KEEP=FALSE;@WAIT=TRUE;))
        )
)

(@RULE=R35
        (@LHS=
                (Yes    (Amp_out_of_alignment))
        )
        (@HYPO=display_AOOA_message)
        (@RHS=
                (Show  ("Amp_out_of_alignment.txt")
(@KEEP=FALSE;@WAIT=TRUE;))
        )
)

(@RULE=R36
        (@LHS=
                (Yes    (Bad_Connection))
        )
        (@HYPO=display_BC_message)
        (@RHS=
                (Show  ("Bad_Connection.txt")(@KEEP=FALSE;@WAIT=TRUE;))
        )
)

(@RULE=R37
        (@LHS=
                (Yes
(Communication_Problem_between_head_Amp_and_low_Pilots_parent))
        )
        (@HYPO=display_CPBHAALPP_message)
        (@RHS=
                (Show
("Communication_Problem_between_head_Amp_and_low_Pilots_parent.txt")
(@KEEP=FALSE;@WAIT=TRUE;))
        )
)

(@RULE=R38
        (@LHS=
                (Yes    (Communication_problem_between_head_amp_and_parent))
        )
        (@HYPO=display_CPBHAAP_message)
```

```
        (@RHS=
                (Show  ("Communication_problem_between_head_amp_and_parent.txt")
(@KEEP=FALSE;@WAIT=TRUE;))
        )
)

(@RULE=R39
        (@LHS=
                (Yes    (Dead_Amplifier))
        )
        (@HYPO=display_DA_message)
        (@RHS=
                (Show  ("Dead_Amplifier.txt")(@KEEP=FALSE;@WAIT=TRUE;))
        )
)

(@RULE=R40
        (@LHS=
                (Yes    (Failure_in_Amplifier))
        )
        (@HYPO=display_FIA_message)
        (@RHS=
                (Show  ("Failure_in_Amplifier.txt")(@KEEP=FALSE;@WAIT=TRUE;))
        )
)

(@RULE=R41
        (@LHS=
                (Yes    (Failure_in_output_of_Head_Amp))
        )
        (@HYPO=display_FIOFHA_message)
        (@RHS=
                (Show  ("Failure_in_output_of_Head_Amp.txt")
(@KEEP=FALSE;@WAIT=TRUE;))
        )
)

(@RULE=R42
        (@LHS=
                (Yes    (Hydro_Outage))
        )
        (@HYPO=display_HO_message)
        (@RHS=
                (Show  ("Hydro_Outage.txt")(@KEEP=FALSE;@WAIT=TRUE;))
        )
)

(@RULE=R43
        (@LHS=
                (Yes    (Line_Break))
        )
```

```
        (@HYPO=display_LB_message)
        (@RHS=
                (Show  ("Line_Break.txt")(@KEEP=FALSE;@WAIT=TRUE;))
        )
)

(@RULE=R44
        (@LHS=
                (Yes    (Nominals_need_to_be_reset))
        )
        (@HYPO=display_NNTBR_message)
        (@RHS=
                (Show  ("Nominals_need_to_be_reset.txt")
(@KEEP=FALSE;@WAIT=TRUE;))
        )
)

(@RULE=R45
        (@LHS=
                (Yes    (Power_Supply_Failure))
        )
        (@HYPO=display_PSF_message)
        (@RHS=
                (Show  ("Power_Supply_Failure.txt")
(@KEEP=FALSE;@WAIT=TRUE;))
        )
)

(@RULE=R46
        (@LHS=
                (Yes    (Unknown_Problem))
        )
        (@HYPO=display_UP_message)
        (@RHS=
                (Show  ("Unknown_Problem.txt")(@KEEP=FALSE;@WAIT=TRUE;))
        )
)

(@RULE=R47
        (@LHS=
                (No     (Downstream_Amp_NOT_Communicating))
        )
        (@HYPO=Downstream_Amp_Communicating)
)

(@RULE=R48
        (@LHS=
                (Yes    (check_Downstream_Amp_Communicating))
                (Execute("match_fault_message")
(@ATOMID=<|topology_data|>.parent;@STRING="match all,\
 without head, any mval, after fault, NO REP";\
```

```
))
        )
        (@HYPO=Downstream_Amp_NOT_Communicating)
)

(@RULE=R49
        (@LHS=
                (Yes    (check_for_Downstream_Effects))
                (Execute("measure_downstream_effects")
(@ATOMID=<|topology_data|>.parent;))
                (>      (downstream_paths.max_affected)(0))
        )
        (@HYPO=Downstream_Effects)
)

(@RULE=R50
        (@LHS=
                (Yes    (at_Terminal_Amp))
                (No     (all_Amps_on_Power_Grid_Effected))
                (Yes    (Amp_Communicating))
                (No     (Warning_Ok_Cycle))
        )
        (@HYPO=Failure_in_Amp)
)

(@RULE=R51
        (@LHS=
                (Yes    (Downstream_Effects))
                (No     (Within_4_Amps_of_Terminal))
                (No     (More_than_4_Downstream_Amps_Effected))
                (No     (Amp_Communicating))
        )
        (@HYPO=Failure_in_Amplifier)
)

(@RULE=R52
        (@LHS=
                (Yes    (Downstream_Effects))
                (No     (Within_4_Amps_of_Terminal))
                (No     (More_than_4_Downstream_Amps_Effected))
                (Yes    (Amp_Communicating))
                (Yes    (Upstream_Amp_Low_Pilot))
        )
        (@HYPO=Failure_in_output_of_Head_Amp)
)

(@RULE=R53
        (@LHS=
                (Yes    (Diagnose_Fault))
        )
        (@HYPO=goto_Head_Amp)
```

```
        (@RHS=
                (Execute("get_head_amp")(@ATOMID=<|topology_data|>.parent;))
        )
)

(@RULE=R54
        (@LHS=
                (Yes    (Downstream_Effects))
                (No     (Within_4_Amps_of_Terminal))
                (Yes    (More_than_4_Downstream_Amps_Effected))
                (Yes    (all_Amps_on_Power_Grid_Effected))
                (No     (Downstream_Amp_Communicating))
        )
        (@HYPO=Hydro_Outage)
)

(@RULE=R58
        (@LHS=
                (Yes    (Downstream_Effects))
                (No     (Within_4_Amps_of_Terminal))
                (Yes    (More_than_4_Downstream_Amps_Effected))
                (Yes    (all_Amps_on_Power_Grid_Effected))
                (No     (Downstream_Amp_Communicating))
        )
        (@HYPO=Line_Break)
)

(@RULE=R57
        (@LHS=
                (Yes    (Downstream_Effects))
                (Yes    (Within_4_Amps_of_Terminal))
                (No     (all_Amps_on_Power_Grid_Effected))
                (No     (Downstream_Amp_Communicating))
        )
        (@HYPO=Line_Break)
)

(@RULE=R56
        (@LHS=
                (Yes    (Downstream_Effects))
                (Yes    (Within_4_Amps_of_Terminal))
                (Yes    (all_Amps_on_Power_Grid_Effected))
                (No     (Downstream_Amp_Communicating))
        )
        (@HYPO=Line_Break)
)

(@RULE=R55
        (@LHS=
                (Yes    (at_Terminal_Amp))
                (No     (all_Amps_on_Power_Grid_Effected))
```

```
                (No     (Amp_Communicating))
        )
        (@HYPO=Line_Break)
)

(@RULE=R59
        (@LHS=
                (Yes    (check_More_than_4_downstream_Amps_Effected))
                (Execute("measure_downstream_effects")
(@WAIT=TRUE;@ATOMID=<|topology_data|>.parent;\
))
                (>      (downstream_paths.max_affected)(4))
        )
        (@HYPO=More_than_4_Downstream_Amps_Effected)
)

(@RULE=R61
        (@LHS=
                (No     (Downstream_Effects))
                (No     (all_Amps_on_Power_Grid_Effected))
                (Yes    (Amp_Communicating))
                (Yes    (Warning_Ok_Cycle))
        )
        (@HYPO=Nominals_need_to_be_reset)
)

(@RULE=R60
        (@LHS=
                (Yes    (at_Terminal_Amp))
                (No     (all_Amps_on_Power_Grid_Effected))
                (Yes    (Amp_Communicating))
                (Yes    (Warning_Ok_Cycle))
        )
        (@HYPO=Nominals_need_to_be_reset)
)

(@RULE=R67
        (@LHS=
                (Yes    (Downstream_Effects))
                (No     (Within_4_Amps_of_Terminal))
                (Yes    (More_than_4_Downstream_Amps_Effected))
                (Yes    (all_Amps_on_Power_Grid_Effected))
                (No     (Downstream_Amp_Communicating))
        )
        (@HYPO=Power_Supply_Failure)
)

(@RULE=R66
        (@LHS=
                (Yes    (Downstream_Effects))
                (No     (Within_4_Amps_of_Terminal))
```

```
                (Yes    (More_than_4_Downstream_Amps_Effected))
                (Yes    (all_Amps_on_Power_Grid_Effected))
                (Yes    (Downstream_Amp_Communicating))
        )
        (@HYPO=Power_Supply_Failure)
)

(@RULE=R65
        (@LHS=
                (Yes    (Downstream_Effects))
                (Yes    (Within_4_Amps_of_Terminal))
                (Yes    (all_Amps_on_Power_Grid_Effected))
                (No     (Downstream_Amp_Communicating))
        )
        (@HYPO=Power_Supply_Failure)
)

(@RULE=R64
        (@LHS=
                (Yes    (Downstream_Effects))
                (Yes    (Within_4_Amps_of_Terminal))
                (Yes    (all_Amps_on_Power_Grid_Effected))
                (Yes    (Downstream_Amp_Communicating))
        )
        (@HYPO=Power_Supply_Failure)
)

(@RULE=R63
        (@LHS=
                (No     (Downstream_Effects))
                (Yes    (all_Amps_on_Power_Grid_Effected))
        )
        (@HYPO=Power_Supply_Failure)
)

(@RULE=R62
        (@LHS=
                (Yes    (at_Terminal_Amp))
                (Yes    (all_Amps_on_Power_Grid_Effected))
        )
        (@HYPO=Power_Supply_Failure)
)

(@RULE=R68
        (@LHS=
                (Yes    (check_Upstream_Amp_Low_Pilot))
        )
        (@HYPO=Set_Upstream_Amp_Low_Pilot)
        (@RHS=
                (Do     ('topol_'\head_amp.name\.parent)(Parent))
        )
```

```
)

(@RULE=R69
	(@LHS=
		(Yes	(Downstream_Effects))
		(No	(Within_4_Amps_of_Terminal))
		(Yes	(More_than_4_Downstream_Amps_Effected))
		(No	(all_Amps_on_Power_Grid_Effected))
	)
	(@HYPO=Unknown_Problem)
	(@RHS=
		(Let	(Power_Supply_Failure)(FALSE))
	)
)

(@RULE=R70
	(@LHS=
		(Yes	(Set_Upstream_Amp_Low_Pilot))
		(=	(Parent.pilot)("Low"))
	)
	(@HYPO=Upstream_Amp_Low_Pilot)
)

(@RULE=R71
	(@LHS=
		(Yes	(check_for_Warning_Ok_Cycle))
		(Yes	(Is_there_a_Warning_OK_Cycle))
	)
	(@HYPO=Warning_Ok_Cycle)
)

(@RULE=R72
	(@LHS=
		(Yes	(check_Within_4_Amps_of_Terminal))
		(Execute("get_distance_to_term")
(@WAIT=TRUE;@ATOMID=head_amp.name;))
		(<=	(amp_to_term.max_dist)(4))
	)
	(@HYPO=Within_4_Amps_of_Terminal)
)

(@GLOBALS=
	@INHVALUP=FALSE;
	@INHVALDOWN=TRUE;
	@INHOBJUP=FALSE;
	@INHOBJDOWN=FALSE;
	@INHCLASSUP=FALSE;
	@INHCLASSDOWN=TRUE;
	@INHBREADTH=TRUE;
	@INHPARENT=FALSE;
	@PWTRUE=TRUE;
```

```
        @PWFALSE=TRUE;
        @PWNOTKNOWN=TRUE;
        @EXHBWRD=TRUE;
        @PTGATES=TRUE;
        @PFACTIONS=TRUE;
        @SOURCESON=TRUE;
        @CACTIONSON=TRUE;
)
```

C.2 Prototype Knowledge Base (prototype.kb)

```
(@VERSION=020)
(@PROPERTY=amp_name@TYPE=String;)
(@PROPERTY=B@TYPE=String;)
(@PROPERTY=C@TYPE=String;)
(@PROPERTY=fault_message@TYPE=String;)
(@PROPERTY=fault_nominal@TYPE=String;)
(@PROPERTY=fault_times@TYPE=String;)
(@PROPERTY=fault_type@TYPE=String;)
(@PROPERTY=fault_value@TYPE=String;)
(@PROPERTY=location@TYPE=String;)
(@PROPERTY=P@TYPE=String;)
(@PROPERTY=parent@TYPE=String;)
(@PROPERTY=pilot@TYPE=String;)
(@PROPERTY=power_grid@TYPE=Integer;)
(@PROPERTY=power_supply@TYPE=String;)
(@PROPERTY=R@TYPE=String;)
(@PROPERTY=RD@TYPE=String;)
(@PROPERTY=SMT_number@TYPE=Integer;)
(@PROPERTY=subs@TYPE=Integer;)
(@PROPERTY=T@TYPE=String;)
(@PROPERTY=termination@TYPE=Integer;)
(@PROPERTY=times@TYPE=String;)
(@PROPERTY=type@TYPE=String;)


(@CLASS=amp_data
        (@PROPERTIES=
                B
                C
                P
                R
                RD
                T
                times
        )
)

(@CLASS=amplifier
        (@PROPERTIES=
```

```
        amp_name
    )
)

(@CLASS=log_data
    (@PROPERTIES=
        fault_message
        fault_nominal
        fault_times
        fault_type
        fault_value
    )
)

(@CLASS=topology_data
    (@PROPERTIES=
        location
        parent
        pilot
        power_grid
        SMT_number
        subs
        termination
        type
    )
)


(@OBJECT=Fault_Amp
    (@PROPERTIES=
        Value  @TYPE=String;
    )
)

(@OBJECT=Fault_Message
    (@PROPERTIES=
        Value  @TYPE=String;
    )
)

(@OBJECT=Fault_Nominal
    (@PROPERTIES=
        Value  @TYPE=String;
    )
)

(@OBJECT=Fault_Time
    (@PROPERTIES=
        Value  @TYPE=Integer;
    )
)
```

```
(@OBJECT=Fault_Type
	(@PROPERTIES=
		Value   @TYPE=String;
	)
)

(@OBJECT=Fault_Value
	(@PROPERTIES=
		Value   @TYPE=String;
	)
)
```

Vita

Surname: Neville Given Names: Stephen William

Place of Birth: Victoria, British Columbia Date of Birth: Feb. 20, 1967.

Educational Institutions Attended:

University of Victoria 1985 to 1992.

Degrees Awarded:

B. Eng. University of Victoria 1990.

Honors and Awards:

NSERC PGS 2 1991-92.
B.C. Science Council G.R.E.A.T. 1991-92.
University of Victoria President's Research Grant 1991-92.

Publications:

Dimopolous, N.J., K.F. Li, A. Watkins, S. Neville, A. Rondogainnis, "An Expert Network Analyzer", *Technical Papers: Canadian Cable Television Association 35th Annual Convention*, Vancouver, B.C., May 31 - June 3, 1992, pp. 123-127.

PARTIAL COPRIGHT LICENSE

Title of Thesis: A Prototype Expert System Based Diagnostic Tool for Cable Trunk Amplifier Networks

Author: ______________________________

Stephen Neville

July 29, 1992.