Office: Technology & People, 5:3 1989) 205-228
Elsevier Science Publishers Ltd, England—Printed in the United Kingdom

Real-time autonomous expert systems in network management*

Rodney M. Goodman, John Miller, Padhraic Smyth
Department of Electrical Engineering, California Institute of Technology, Caltech 116-81, Pasadena, CA 91125 (U.S.A.)

Hayes Latin
Information Systems Organization, Pacific Bell, Concord, CA 94520 (U.S.A.)

ABSTRACT

In this paper we describe our approach to developing real-time expert systems for integrated network management. Such systems are becoming urgently needed as the complexity of networks and the pace of new technology increases. Real-time expert systems are very demanding and require interfaces to existing databases and sense and status systems. There are thus significant integration issues in their design. In the paper we first introduce expert system technology, and then develop the requirements for a real-time system in the network management domain. We then describe the integrated approach to network management being developed by Pacific Bell, and show how expert systems fit into this model. Finally we describe a prototype real-time expert system, NETREX, which is aimed at automatically maintaining Pacific Bell internal data networks. In particular, we describe a prototype system that implements one aspect of NETREX, a trouble ticket expert. The prototype scans open (active) trouble tickets in real time and then proceeds to collect sense and status information from the network environment. A diagnosis is made, annotations are made to the ticket, and a repair is attempted if possible.

1. INTRODUCTION

Large data networks are becoming unmanageable with human effort alone. Such networks are often widely geographically distributed and consist of many thousands of terminal devices. Switching centers can consist of dozens of nodes with numerous different vendor equipments and network protocols. The advent of LANs and the prospect of ISDN only makes the problem worse. In addition, practical network

* Paper presented at the First IFIP International Symposium on Integrated Network Management, Boston, 14-17 May 1989.

evolution is rarely planned but is implemented on an ad hoc basis as user demand dictates. The task of fault diagnosis, isolation, and repair in such an environment falls to the network management operation, and is very demanding in terms of human expertise. This expertise is limited by both lack of skilled personnel and the rapid pace of technological development. The goal of any network management operation is to reduce network downtime and operating costs. Expert systems can help to achieve this goal.

Expert systems have been used in other fields to perform fault diagnosis, simulation, testing and monitoring, intelligent user interfaces, planning tools, and other decision-intensive tasks in order to relieve the shortage of skilled staff. The technology is now proving an appropriate technology to tackle problems in network operations and management. The use of expert systems in network management has recently mushroomed [1, 2]. The types of application reported can roughly be split into two types: those concerned with network simulation, planning, and design; and those that monitor, control, interpret, diagnose, and repair networks. Examples of the former include: KAT—an analysis tool for network planning [3], ENS—an aid for the design and sales of networks [4], DESIGNet—a tool for network design and modeling [5], and SIMNETMAN—a simulation and design aid for packet switched networks [6]. Those in the second category include what must now be classified as a 'famous' expert system, ACE [7]. Automated Cable Analysis (ACE) was one of the first expert systems to be used in telecommunications, and is one of the few in production today. ACE operates routinely at night in several telephone companies, producing expert-level diagnoses on cable faults. Other network diagnostic systems include NDS—a diagnostic system for nationwide communications [8], Troubleshooter—an AT&T system that diagnoses troubles in Datakit VCS networks [9], and ESTA—a system for analyzing trunk outage codes [10]. Systems concerned with network monitoring and control include: NEMESYS—a system that displays congestion in the AT&T long-distance network [11], and DAD—a BNR system that monitors the Canadian packet network [12]. Expert systems concerned with testing, maintenance, and repair include: COMPASS—a system for CO maintenance of the GTE No. 2 EAX switch [13], MAD—a maintenance advisor for the DMS family of switches [14], and SMART—a maintenance and repair tool for the AT&T 1A ESS switch [15]. In addition, new systems are constantly appearing.

Most current expert systems operate in a 'consultant' paradigm. That is, a user sits down at a terminal and goes through a question and answer 'consultation session' with the expert system (ES), and receives some advice on the problem in hand. This paradigm is appropriate for the first category of tasks, which are concerned with planning, simulation, and design. However, the real promise of expert systems in network management lies in expert systems that can autonomously go about their tasks in real time. Such systems must be able autonomously to monitor and control the network in real time, interact with databases, and instigate repair action on their own. The success of the ACE system is due in no small part to its interaction with the conventional cable repair database, and its ability to monitor and test its environment automatically.

In this paper we are concerned with autonomous real-time diagnostic and repair expert systems. As yet, very few such systems are in operation in any problem domain, although components of the real-time aspect are in place. Example systems include: YES/MVS [16], REACTOR [17], and VM [18]. Even fewer perform diagnosis of communications systems [19, 20]. This is due in part to the pervading 'consultant' paradigm found in expert system software, and the 'island of technology' implementation platforms, such as LISP machines. To realize the potential of expert systems in network management we need to integrate AI and expert systems methodology with existing software and hardware techniques. Fast interfaces to existing software modules are required, as well as a change of mind set away from the 'consultant' paradigm. In a real trouble analysis center (and in many other real-time decision-making organizations) such a paradigm is totally impractical. The ES users do not have the patience to answer the expert system's questioning as it tries to formulate a model of the problem—they want to be solving the problem. In addition the proposed users (the network trouble and repair staff) generally have a high level of skill, and will not tolerate questioning at a lower skill level—and this level varies from person to person. Thus the ES must work in parallel with the human system, and acquire its own picture of the problem environment. Having done so, the autonomous system can act in two possible ways. First, the ES can instigate its own problem repair completely autonomously, without consulting the human experts. Alternatively, the ES can interact with the human experts. If the latter course is taken, the ES must interact at a sufficiently 'high' level if it is to gain acceptance by the human experts. The ES must therefore be integrated into the existing human-operated management system with suitable smart user interfaces. In order for an autonomous system to be possible in the network management environment, there has to be in place an 'effector' and 'sensor' system with which the ES can observe and manipulate the network. Also, access to network component and topology databases is needed. It is vital that this system is in place and relatively stable, otherwise the ES design will devolve into this lower level design effort, as opposed to the higher level design effort needed for expert-level problem solving. This situation mirrors that of early computerization, where a task that had an existing manual system could be computerized efficiently and elegantly; whereas one that did not took much more effort and hence cost.

In this paper we outline our approach to designing comprehensive real-time autonomous expert systems for integrated network management. Our practical experience comes from a collaborative effort between Caltech and Pacific Bell (P*B), in which the mix of research and real-world environments has been very successful in tackling this highly demanding and hard problem domain. Thus our approach is intimately linked with the databases, and effector-sensor systems being put in place by Pacific Bell. However, many of the problems we have tackled are generic to the network management scenario, and even exist in many domains outside network management.

In the paper we describe ISM (Integrated System Management), a Pacific Bell solution to the network management problem. ISM consists of three subsystems:

NETREX, LINC and TIMS. The first subsystem, NETREX (NEtwork Trouble and Repair Expert), is a high-level collection of controlling expert systems which uses the other two subsystems to interact with the environment. LINC is the sensor and effector system which allows remote access to any network device. TIMS is the network inventory and trouble-reporting database system. These two subsystems are substantial pieces of software in their own right, and have taken many years to develop. Without them NETREX would not be feasible. We briefly describe these systems in the paper, but of necessity we concentrate on the expert systems design.

The paper develops as follows. First, we give a brief overview of expert system methodology. We then formulate the expert system requirements needed in the network management domain. We then outline the problem domain, using our experience in the Pacific Bell network management environment. This leads us to describe the basic design philosophy for ISM and NETREX, and their real-time operation. We then describe one of the prototype NETREX modules—a trouble ticket and repair expert. Finally, we describe the ongoing facets of NETREX, and draw some conclusions from our experience so far.

2. EXPERT SYSTEMS

In this section we give a brief overview of expert system technology.

An expert system is an example of a *knowledge-based system*. Such systems are concerned with the acquisition, organization, and representation of knowledge, and the identification and use of heuristics and inference techniques to reduce the complexity of problem solving in a given domain.

A knowledge-based expert system is simply a problem solving program that uses knowledge derived from both education by the *domain expert* in the form of *rules* and *relationships*, and from *experience* (past examples). The key idea in expert systems is to separate the domain knowledge into a *knowledge base* distinct from the procedural processes which manipulate that knowledge. A conventional program, on the other hand, buries these two in code elements. An operating ES consists of a knowledge base together with an *inference engine, working memory*, and *user interface*. The knowledge base (KB) contains the facts, ideas, relationships, and rules that specify the problem domain. The inference engine uses the KB to infer new facts, analyze situations, hypothesize, draw conclusions, and initiate actions. The working memory contains the current status or knowledge of data instances. The user interface supports the *consultation paradigm* being used to solve problems in the domain. The program can also explain its reasoning. The ES program is built using an *expert system building tool*; that is, a program development environment that supports interactive debugging of program code, knowledge and data. The *expert* is one or more persons skilled at solving problems in the domain, and the ES models the experts' problem-solving techniques. Finally, the *knowledge engineer* (KE) ties all these parts together. The KE knows how to build expert systems, interviews the experts, organizes the knowledge and its representation within the expert system, and may write much of the program code. Expert systems are clearly different from conventional procedural software systems, but they share the same requirements of rigorous software design methodology. Key among these is the process of knowledge acquisition and representation.

Knowledge representation is one of the knowledge engineer's main tasks. A given problem may require one or more representations, and conversely the expert system may be severely hampered in both speed and flexibility if the representation scheme does not fit the problem domain. We now briefly examine these because of their critical importance in the overall ES design.

Propositional logic is the logic of Boolean algebra and digital circuit design. It is limited because of the requirement for logical completeness in a factual or problem statement; that is, all alternatives must be considered—and this is too broad for real-world problems. Small fragments of knowledge may be adequately expressed in logic. *Predicate logic* is an extension of propositional logic which allows richer statements (*predicates*) about real world *objects* (arguments). This logic forms the basis of the AI programming language PROLOG.

Semantic networks are collections of *nodes* (objects) and *links*, which can be relations, predicates, or actions. The network depicts hierarchic relations between objects and allows for *inheritance* of object attributes by other objects. For example, the object **line** is hierarchically linked to the object **node** and thus can inherit attributes of **node** without any explicit duplication of data. The network can be used for inference as well as representation. This representation is an example of an *object-oriented representation* and is extremely flexible. The inheritance aspects allow much redundancy to be eliminated in the representation, and allow reasoning to occur at different hierarchic planes.

Simplified hierarchies are used in expert systems to constrain the too flexible semantic net. Most popular are *object-attribute-value* (OAV) triples, trees, and lists. All of these methods allow for inheritance to a degree and OAV triples are used in many expert systems. For example, a P*B network is an *object* that may have an *attribute* called switched which can have the value true or false. Hence POSN/A is an *instance* of the class *network* and its *switched* attribute is true.

Frames expand on the OAV representation to provide a richer representation environment. Frames can combine procedural and declarative knowledge and are naturally object oriented with inheritance. They represent the most powerful representation scheme used in expert system building tools today. Frames contain *slots* for characteristics and attributes. The slots contain names of attributes with values (possibly default), pointers to more detailed frames (subclasses), or pointers to *rules* or *procedures* for calculating values. For example, we may define a User Terminal Frame which belongs to the class of terminal equipment, and has attributes such as device ID, line number, pointer to control unit slot, baud rate demon, etc. The baud rate demon will invoke a procedure to status the device and return the current baud rate. The pointer to the control unit (CU) slot can be used to inherit the attribute "is_broken" if the CU to which the terminal is connected is down.

Production rules are one of the most popular methods for capturing knowledge in small human-size chunks. They are naturally self-documenting and fit into the way experts verbalize their knowledge. They take the form of IF–THEN rules in which one or more *antecedents* are linked together with the logical AND statement, such that if all the antecedents are true then the *consequent* is presumed true. For example,

IF the terminal ID corresponds to EU0 in the THP,
AND EU0 is down
THEN the user cannot access applications

Rules can be incorporated into the previous knowledge structures with ease. For example, EU0 is an *object* which has an *attribute* "is_down" which has the *value* FALSE.

Inferencing is supported by the *inference engine*, which uses both the knowledge representation scheme and the production rules to solve problems. The engine operates on a recognize–act cycle; facts are recognized, rules are fired or hierarchic/semantic inferences are made, and new facts emerge which are put into working memory. The use of both rules and structural models of the problem domain is known as *causal* or *deep reasoning*. Such systems are capable of much more 'intelligent' behavior than rules only (*surface*) reasoning and *model-based reasoning* is a commonly used variant. The inference mechanism is concerned with both the *mode* of inference, and the *control* of inference. The most usual type of inference is common sense (*modus ponens*) inference; that is, a rule such as IF A THEN B means that if A is true we expect B to be true. Also of major practical importance is inferencing with uncertainty. Control aspects include *forward/backward chaining*, the *depth* and *breadth* of the search (how focused or greedy it is), how to resolve conflicts in inferred outcomes, and how to deal with facts that change during consultation (monotonicity).

Forward chaining is an inference control method which works forward from known database facts, or user queried facts to a conclusion or hypothesis. The inference engine attempts to work forward by satisfying the antecedents of a rule; it is thus *data driven*. One of the matching rules is selected to be *fired*, by means of priorities or *meta-rules* (rules on how to select rules). The inferred consequent can then be added to the known facts in working memory. The cycle then repeats until a conclusion state is inferred or no more rules fire. Thus in this mode we usually wish to predict an outcome given a set of known conditions or symptoms. For example, we may wish to conclude a network repair scenario from the results of status or other measurements.

Backward chaining starts from a goal, conclusion or hypothesis, and then searches for supporting evidence. This is the mode in which we are presented with an outcome (e.g. the user's terminal does not work) and we wish to work back to the possible causes or symptoms. The inference engine has to satisfy intermediate goals on its way backwards. For example, a rule may say IF the user's CU is down THEN the user's terminal is down. We now have a subgoal of CU_down to satisfy. Another rule may be IF the EU is down THEN the CU is down. Thus if the EU is in fact down we can infer that this is the cause of the terminal being down. In practice a combination of both backward and forward chaining is needed for sophisticated inference.

There are various aspects to control that may be required for a particular problem domain. Firstly, rule firing can be depth or breadth first. If for example a rule fires and we immediately set about trying to use the consequent we are performing a depth-first, highly focused, or greedy search. We may choose to collect all rules that fire on a particular cycle and then choose the 'best' to fire—this is breadth first. Or we may mix the two. Given a number of rules ready to fire, how do we choose between them? This may be done on the basis of priority or importance values, confidence values, recency—suppress recently fired rules, or specificity—the greater the number of antecedents in a rule the more specific and 'focused' it is. These aspects of expert systems are used but their effect is not well understood as yet.

The other major area of practical importance, not well developed, is that of using confidence values and handling uncertain or inexact data. Both facts and rules can be uncertain or unknown, and *unknown* should always be an allowable response from a user. Rules should propagate uncertainties from antecedents to consequents. Similarly, rules themselves can have confidence values associated with them,

IF EU0 is_down **THEN** EU1 will overload (with confidence 70%)

Also confidences can be used in the control process to fire rules, e.g.

IF problem_is_hardware with confidence >90%

THEN dispatch_fix-it_agency

The area of confidence and its use is the least well understood aspect of expert systems at present and is the subject of ongoing research. However, because certainty is so close to human reasoning it should always be built into the expert system.

Hypothetical reasoning refers to solution approaches in which assumptions may have to be made to allow the search to proceed. Later on, however, certain assumptions may turn out to be invalid and may need to be retracted. This *non-monotonic reasoning* is handled in several ways. One approach is to carry along multiple solutions (hypotheses) in parallel, and discard the inappropriate ones as contradictory evidence is accumulated. This is referred to as *viewpoints*, *contexts* and *worlds* in different tools. Another approach is to keep track of the assumptions that support the current search path and to backtrack to the appropriate branch if the current path's assumptions (truth) is contradicted. This approach is called *truth maintenance* or non-chronological backtracking.

3. EXPERT SYSTEM REQUIREMENTS IN THE NETWORK DOMAIN

In this section we examine the system requirements needed to implement real-time expert systems in the network management domain. The entire AI armamentarium is needed in this domain to implement the desired characteristics of uncertain reasoning, temporal reasoning, model-based reasoning, non-monotonicity, and automated learning.

Clearly a real-time system needs to keep pace with significant network events. The fastest such events are 'alarm' events, and the slowest are operator interventions. All occur asynchronously and unpredictably. Given an alarms-driven system, this could mean events being generated in periods of the order of a minute. Also the system must be able to have a strategy for handling an avalanche of events such as those caused by node failure without simply giving up. It may be necessary to build an intermediate representation to handle such frequent events. Thus a 'local' ES can perform useful temporal reasoning before passing the new event on to a higher 'global' ES with greater (structural) reasoning power.

The ES must maintain several accurate models of the network. These models will be used for deciding on appropriate test strategies given the particular devices involved, and also for model-based reasoning in the inference process. The ES needs a model of network connectivity at the abstract or functional level (nodes are connected by links, etc.) as well as a device configuration model (e.g. the device at location xxx is of model yyy). Furthermore, the ES must maintain an accurate genning model, that is, the current 'static' connectivity as believed by the network operators. To preserve the accuracy of these models the ES must periodically update the models by several means. These include reference to the network operator's genning database, and autonomous probing of the network. In this way the ES can 'learn' the specific connectivity of the network. It can also spot genning mistakes (e.g. "this device is genned as a terminal but I get a printer response code from it"). There is an inherent trade-off between the freshness of the data in the models and the access time required to update. However, the ES in the network management situation has plenty of time at night to perform these housekeeping tasks.

The ES needs to integrate the heuristic knowledge of the network as used by the network operators to detect, diagnose, and repair faults, with the structural models being maintained. The type of knowledge representation becomes a key issue here. A rule-based representation is best for the heuristic knowledge. The structural knowledge can be implemented in several ways. An object-oriented or frames-with-inheritance representation is most appropriate, given the usefulness of inheritance in the network domain (e.g. a line can inherit the 'down' characteristics of a node if the node is down). Object-oriented representations easily map onto the network devices. Such representations turn the network objects into 'smart' objects that understand the types of operations allowable on themselves. This may or may not be desirable. In addition, it may be necessary to integrate the representation with an existing network database. In this case if the network is large then replicating this model within the ES is just not feasible. In this case an integration issue arises between the external database and the ES. However, many of the benefits of the object representation can be retained by mapping the database (provided it is a relational database) onto on object framework. The essential requirement is that there should be a mechanism for defining classes, superclasses, instances, and inheritance, because these elements are so close to the physical network itself. Given that class inheritance can be implemented the representation in this case can range from simple object-attribute-value to a full frames model.

The ES must be able to deal with multiple problem instances. As opposed to the consultation paradigm, there are usually several problems that are coexisting in a large computer network. The ES must be able to look for related problems and to determine causal relations between problems. This may dictate that the ES has to suspend diagnosis of a particular problem and switch to a related problem, particularly in the case when one problem subsumes another. For example, the ES has to be able to recognize that it is no good dealing with an influx of 'line-down' alarm-generated problems, when it has a front-end module alarm problem which subsumes all the line problems. This implies that there has to be some prioritizing of problems. In the case of networks this can fairly naturally follow the hierarchies in the network models. However, problems on equal levels may have different priorities and require immediate escalation. For example, service outages in the billing departments may have perceived greater cost than outages in other departments. If problems are to be suspended and resumed, data freshness becomes an issue, and a resumed problem may have to acquire new status data. This is particularly true if the resumed problem has in fact been solved by a previous problem diagnosis.

Related to the above is the requirement that the ES must reason with temporal data and be aware of non-monotonicity. Status can easily change during diagnosis and in fact the problem can often 'come clear' spontaneously or as a result of performing tests. The inferencing strategy must be able to support revision of belief as the diagnosis proceeds. The ES must also have a formalism for dealing with recurring, intermittent, and chronic faults. There must thus be further abstract models of the domain that deal with temporal correlations (particularly for alarms data) as well as the spatial models outlined before.

The ES must be able to reason with uncertain and incomplete data. Very often an incomplete and uncertain picture of the problem is developed. For example, information can be missed off trouble tickets, or alarm events may not be gathered owing to the particular type of failure itself. The ES must also use probabilistic data on component failure. The latter can be gathered from network operator heuristics as well as failure database records. The particular type of uncertainty model does not matter (in fact, this area is still very much a research topic)—the important point is that probability and belief estimates should be built into the reasoning structure from the beginning because it is so appropriate to the network management problem.

The ES must be able to instigate tests autonomously and to assess the results of these tests. These tests have a cost function in terms of the time taken to perform them and this needs to be traded off with the usefulness of the test. For example, a costly test may involve taking some in-service lines out for the period of the test, but if this is the only way to establish the cause of some hard intermittent problem there could be significant payoff.

The ES must present the results of its diagnoses in an appropriate manner. This may involve different user interfaces, and degreees of autonomy. This requirement is related to the testing of the expert system. To implement such complex systems it is not usually possible to have a 'cut-over' to an ES-controlled system. Unlike procedural programs, expert systems can be wrong and still be correct. An ES diagnosis may be incorrect just as a human one may be. Thus the ES must usually be implemented in an incremental manner; that is, starting by passively shadowing the human system, and acquiring more pro-activity and autonomy as confidence increases in its conclusions and actions.

Many of the above requirements tax the latest expert system development shells and practices. At Caltech we are researching into new expert system methodologies which will address many of these problems in this and other domains. We have proposed a new and novel information theoretic approach to expert systems design. In particular we are using the well established principles of information and communications theory to propose new methods of expert system design [21–34]:

- automatic derivation of decision trees and generalized rule graphs from real-world example data;
- handling uncertain information and how uncertainty propagates through inference (uncertainty calculus);
- knowledge representation from examples;
- controlling and guiding the inference process;
- developing neural network inference systems for extremely fast parallel non-monotonic inference.

Our new approach is as follows. We first developed the requirements of a *consistent* approach to probabilistic rule-based expert systems, which can start to overcome current problems. This led us to propose a *probabilistic production rule*, and define a measure (the J-measure) of the information content (or goodness) of the rule. We next tackled the problem of *learning* rules. We have shown how our J-measure has many desirable features for automated rule induction and has a natural interpretation from a cognitive science viewpoint. In particular we have developed an algorithm (the ITRULE algorithm) which operates on example data to produce a set of the most informative rules *automatically*. This led us on to propose the use of the J-measure in goal directed *probabilistic* inference, conflict resolution, and control. We have shown that the information content of a rule can be used to implement locally optimum backward- and forward-chaining strategies, which leads to a context-dependent decision scheme. We then developed techniques that map our probabilistic rules onto a neural network to achieve fast parallel inferencing. Our current research is directed at integrating these techniques into a coherent expert system methodology that leads from real-world sensory data all the way through to neural network run-time execution.

Several of these topics are very appropriate to the network management problem. For example, our efficient decision tree design methods are a natural way of specifying the rules for fault finding, repair, classification and diagnosis procedures. These algorithms enable very efficient decision trees to be derived from past example data, even under the presence of large amounts of noisy (uncertain) data. We have also developed generalized rule finding algorithms for deriving rules from examples stored in databases. This is being used to find rules from alarms data (where no real expert exists), and to assess the operation of the expert system modules by developing meta-rules between the expert system's diagnosis and the final human audit. Our work on rule-based neural network inferencing architectures has potential for fast non-monotonic expert systems that can learn dynamically from the network environment. We cannot go into the detail of these techniques in this paper. However, Fig. 1 shows an example of automatically deriving rules from the alarms database. Such rules can be used to relieve the knowledge acquisition bottleneck to a certain extent in this case. Because network management is now overwhelmed with alarms, no human really analyzes them. The rules derived by ITRULE can be used to show experts possible correlations between alarms and encourage them to formulate rules that would be very difficult to acquire otherwise. In this sense the rule output of ITRULE forms a digestible executive summary of huge amounts of raw data.

4. THE PROBLEM DOMAIN

In this section we outline a typical network management problem domain. Of course, we deal with the actual environment as it exists in our Pacific Bell collaborative project. However, this domain is very similar to other network management operations, and indeed the domain is similar to many other non-telecommunications trouble management centers.

The problem domain we are dealing with is the management of P*B internal data networks. These networks support vital company applications such as billing, service orders, inventory and so forth. The efficient operation of these networks and the provision of very high up-time is of vital commercial value to the company. The hosts that support these applications are distributed in several major computer centers over California; each computer center contains hundreds of mainframe hosts. The networks that support these applications are varied (e.g. async, bysync, packet, LAN, etc.) and large (e.g. 20 000 terminals on the BANCS bysync network).

The task of managing the networks and providing the system of trouble analysis and resolution takes place in several Trouble Analysis Centers (TACs). The software that supports this activity is ISM (Integrated System Management). The ISM software consists of three integrated parts, NETREX, LINC, and TIMS. NETREX is the expert system manager which uses the two external programs LINC and TIMS to obtain information on the network environment. The LINC program provides the means to remotely manipulate and sense the network (the effector and sensor system). TIMS is a database that stores inventory, genning information, and supports the trouble ticket and alarms log infrastructure of the fault isolation and repair process. These two modules are very significant pieces of software in their own right and without them it would be impossible to implement the level of real-time expert system we envision.

Consider the manual system as currently operated by the TACs. A client with a problem calls the center and explains the problem to a data specialist or 'call

ITRULE

ITRULE is being used with experts to:

* Identify topological rules:

  IF control__unit a is down
  THEN terminals xaa down

* Learn repair scripts:

  IF terminal__status = fail
  AND control__unit **NOT** down
  THEN disable/enable line

* Identify temporal loadings:

  IF time = (>9.30 am **AND** <10.30 pm)
  AND NODE 1 = alarm
  THEN NODE 2 = alarm in 15 minutes with prob = 0.8

* Analyze Trouble tickets:

NODE	STN	Dev type	SYMPTOM	DEV S	DEV no	AL ST	RC CU	IL XIL	DL XD	BCT	Ref1	Ref 2	Close Cat	Soln Dev	Soln Act	Soln By	Fix By	time down
D0	40	4540A	?	CUF	ALLDEV	•	OK	•	•	•	•	•	TERM	CU	RC	CS	CLIENT	under1hr
D0	C1	TC174	?	CFL	ONE DEV	•	OK	•	•	•	•	•	TERM	CU	RC	CS	CS	under1hr
B0	C4	4540A	NO CO	•	•	OK	NOK	•	•	•	PCO	•	NO TRBL	•	•	•	•	over3hrs
B0	C1	TC174	NO CO	CUF	?	•	•	•	•	•	PCO	•	NO TRBL	•	•	•	•	over3hrs
B0	40	4540B	NO CO	•	•	•	•	•	•	•	PCO	•	NO TRBL	•	•	•	•	under3hrs
B0	40	TC174	NDE DWN	CUF	ALLDEV	•	•	•	•	•	TESTER	•	TELCO	LINE	?	?	?	under1hr
B0	40	4540A	?	CUF	ALLDEV	•	NOK	NOK	•	•	TESTER	OCS	TELCO	LINE	?	TESTER	PCO	over3hrs
B0	40	SCC	?	CUF	ALLDEV	•	•	•	•	•	TESTER	•	NO TRBL	?	•	•	•	under3hrs
B0	C1	4540A	?	CUF	ALLDEV	•	•	•	•	?	OCS	•	TERM	CU	HRDWR	OCS	OCS	over3hrs
B0	C3	4540A	S/R	?	?	•	NOK	•	•	NOK	TESTER	•	TCU SWR	CU	SFTWR	TESTER	TESTER	under1hr
B0	C1	TC174	INTRMCU	?	?	•	•	•	•	•	OCS	•	TERM	D/S	LOOPBK	OCS	OCS	under3hrs
B0	40	TC174	?	CUF	ONE DEV	•	•	•	•	•	OCS	•	DATA SET	D/S	HRDWR	OCS	OCS	over3hrs
B0	40	TC174	INTRMCU	•	•	•	•	•	•	•	OCS	•	TERM	CU	HRDWR	OCS	OCS	over3hrs
B0	C1	4540A	?	CUF	ALLDEV	NOK	•	NOK	?	•	TESTER	•	DATA SET	D/S	CONFG	TESTER	CLIENT	under1hr
F0	C1	4540A	?	CUF	ALLDEV	•	NOK	NOK	•	•	PCO	•	TELCO	TCXR	HRDWR	PCO	PCO	over3hrs
D0	40	4540A	ALL DWN	CFL	SOME DEV	•	NOK	•	OK	•	TESTER	•	TCU SWR	LINE	SFTWR	TESTER	TESTER	under1hr
D0	40	3274	TRM DWN	•	•	•	•	•	•	•	OCS	•	TERM	CU	HRDWR	TESTER	OCS	under1hr
F0	40	554612	?	LMF	•	•	•	•	•	•	TESTER	•	TCU SWR	LINE MOD	SFTWR	TESTER	TESTER	under1hr
F0	40	4540A	?	CUF	ALLDEV	•	•	•	•	•	TESTER	•	TERM	CU	HRDWR	OCS	OCS	under1hr
F0	40	4540A	ALL DWN	•	•	•	NOK	OK	OK	•	TESTER	•	TCU SWR	LINE	SFTWR	TESTER	TESTER	under1hr
F0	40	4540A	ALL DWN	CUF	ALLDEV	•	NOK	NOK	•	•	TESTER	•	NO TRBL	LINE	•	•	•	under1hr
F0	40	4540A	S/R	CUF	ONE DEV	•	OK	•	•	NOK	TESTER	•	MUX	MUX	RESET	TESTER	NCG	under1hr

Rules:

	IF	THEN	P				J
1	Solved__By OCS	Fixed__By OCS	1.000	0.308	0.346	1.53051	0.47093
2	Close__Cat NO__TRBL	Fixed__By *	1.000	0.192	0.192	2.37851	0.45741
3	Soln__Actn SFTWR	Fixed__By TESTER	1.000	0.154	0.154	2.70044	0.41545
4	Close__Cat TCU__SWR	Fixed__By TESTER	1.000	0.154	0.154	2.70044	0.41545
5	Fixed__By OCS	Solved__By OCS	0.889	0.346	0.308	1.06719	0.36941
6	Fixed__By PCO	Close__Cat TELCO	1.000	0.115	0.154	2.70044	0.31159
7	Soln__Actn RC	Solved__By CS	1.000	0.077	0.077	3.70044	0.28465
8	Fixed__By OCS	Soln__Actn HRDWR	0.889	0.346	0.385	0.79991	0.27689

Fig. 1.

screener.' This is a task that involves interaction with the client and many vague problems are cleared up at this point. This task is absolutely not suited to an expert system because of the breadth and shallowness of the problem domain; for example, anything from the power cord not plugged in, to coffee stains on the keyboard. If the problem is non-trivial a trouble ticket is entered into the TIMS database. The data specialist may (depending on expertise) then try to resolve the problem by doing more tests. If the problem is resolved, a client premise 'fixit' agency may be dispatched. If the trouble cannot be resolved at this stage more sophisticated testing is necessary, and the data specialist passes the ticket on to the Comtech section. In parallel with this a 'statuser' monitors open trouble tickets and commit times to ensure proper prioritizing of the tickets and to deal with client interaction. Comtech personnel are very experienced in data communications and have a deep understanding of the topology of the network. Also, Comtechs have more sophisticated network access and data scoping equipment with which to probe the network. For example, the ANMACS system is a sensor and effector system accessible from LINC which enables data lines anywhere in the state to be broken for loopback and scoping tests. Also, automatic alarms are continually being logged by the TIMS/LINC system and displays of these are continuously available to the Comtech section. Using these tools the Comtechs can quickly dispense with problems of a medium level, which may have been out of the capability of the data specialists. This leaves the really hard problems such as intermittent faults, or those that require significant client–fixit agent interaction, and these may take considerable expert-level diagnosis to solve. Finally, the problem may be resolved, in which case the trouble ticket is closed with the appropriate annotations; or the responsibility for the ticket is passed to another agency (e.g. plant control, or customer premise equipment).

It is clear that automation of the above procedures would result in significant benefits in terms of faster problem resolution and hence shorter client down-time. Current problems that occur in the manual system are as follows. First, no preventative maintenance is done. The TAC is too busy fire-fighting in the day to do such work, and only a skeleton staff exists at night. Secondly, many problems are passed on to the Comtechs that are really low-level problems. This occurs because of the variable skill levels in the data specialist section. Also, many problems prove to be transient and simply come clear. Thus, valuable expert time is spent on relatively trivial problems. Thirdly, the alarm displays provide too much information for the Comtechs to use effectively (the three mile island problem), thus valuable alarm information that could help problem resolution in difficult cases (e.g. intermittent faults) is often ignored. Furthermore, automated alarm-driven diagnosis would enable problems to be resolved before they even resulted in a trouble ticket, and thus also save valuable Comtech time.

5. EXPERT SYSTEM SCENARIOS

The P*B ISM system is an approach aimed at solving the domain problems outlined in the last section. The ISM system is an integration of NETREX, TIMS, and LINC.

In this section we outline what we envision the final form of NETREX to be. NETREX is envisioned as a collection of expert system modules sharing knowledge bases, databases, and network manipulation methods in order to provide a number of specific problem solution tools and methodologies. A fundamental assumption is that NETREX works and interacts with users and other software systems in real time. Furthermore, NETREX modules do not operate in a 'consultant' paradigm in the traditional sense, but rather have a high degree of autonomy.

The types of NETREX modules can be split into two types: those that are completely autonomous, and those that interact with a human user. First, we consider those modules that interact with a human user.

The *data specialist's assistant* is one of NETREX's modules working in the background of the data specialist's terminal. As soon as a trouble ticket is opened NETREX starts its own data gathering and analysis of the problem. NETREX 'shadows' the data specialist and offers relevant data and its own problem analysis, if appropriate—for example, if there were any automatic alarms generated on this device or its peers in the hierarchy. Or previous trouble reports and their resolution. For example, NETREX can quickly say, "The client's screen is dead because I have an alarm that 20 minutes ago the processor hosting the application went down."

The *Comtech workstation* is a development of the above assistant, which would aim to integrate all the scoping and network management tools into a fully windowed environment. The module would interact with the Comtech at a high level, and be able to answer complex hypotheses. For example, "What will be the effect on site X if I take node 3 out of service now?" Such impact analysis could have applications in load planning and network analysis.

The *field repair assistant* would interact with field repair agencies as they try to fix problems at customer premises. The module would allow network manipulation and test without tying up the skilled Comtechs.

The second type of module is the type that operates without human intervention. The objective here is to automate the network problem resolution and repair process as far as possible.

The *alarms monitoring and repair expert* is perhaps the most important and real-time-intensive operation to be attempted. Alarms from all network components are being continuously monitored and stored by the LINC process. Alarms would be filtered and passed to the alarms expert in real time for processing. Raw alarms would be processed into significant events, some of which would be simple, such as indications of processor overload, to a sophisticated analysis of a recurring and intermittent trouble. Problem resolution would also be undertaken by this module. Repair scripts would be automatically evoked to clear the trouble. These would range in complexity from simple restarts of network components to complex sequences needed to bring up failed network hosts. Becoming alarms driven brings both benefits and problems. The expert system must now be truly real-time in a machine time sense, as opposed to a human real-time system.

The *trouble ticket expert* module would be positioned between the data specialists and the Comtechs. The objective would be to remove trouble tickets from the stack being queued for Comtech attention. If the problem can be cleared by the ES the ticket is removed from the stack, again saving valuable Comtech time. The module would also perform statuser and escalator checking of pending trouble tickets, making sure tickets are not 'forgotten.' This is particularly important when responsibility for the ticket is shared among various agencies.

The *network status and configuration monitor* has responsibility for maintaining an accurate model of the network in the face of changes due to genning new equipments into the network, as well as outages. The module would do automatic health monitoring during the night. The health monitor may automatically produce a trouble ticket. A desirable characteristic of this module would be the ability to learn about the operation and configuration of the network automatically. This information would be of use in impact analysis.

Knowledge extraction, auditing, and learning is an autonomous module which tries to derive new rules and knowledge for future use by the other expert systems. A long-term view is taken, and the system works with the historical data in the database. The ES uses examples in the form of past trouble histories, repair times and procedures, etc. The system would automatically monitor and audit the success rates of the other expert systems. It could help to identify intermittent faults. It could correlate seemingly unrelated actions or events to produce new knowledge. Finally, such a system can identify the features or attributes that are most important in influencing or causing certain outcomes; for example, the top ten events that influence down-time. Because of the real-time nature of the domain it should be possible to collect large sample statistics. As the ES evolves it can self-modify its rule base and achieve continuous improvement in performance. Subtle but powerful types of rules not obvious to the experts can be detected, and in particular the rule certainties (which may have been only roughly guessed at by the experts in the beginning) can be refined for optimal performance. The knowledge produced from such data is of great commercial value.

6. NETREX DESIGN PHILOSOPHY

ISM is an integration of the NETREX, LINC, and TIMS modules. There are thus significant interface and integration issues which are being tackled as part of our overall design. This integration is vital in order to achieve real-time operation of the expert system scenarios outlined earlier.

The LINC (Localized Intelligent Network Control) system as implemented by Pacific Bell provides an interface to a variety of data communications systems used at various geographic locations within the company. LINC allows operators to interact with remote sites for testing alarms monitoring and log message management. Custom user interfaces allow operators to perform network commands for component testing and isolation with simple LINC syntax. Thus operators do not have to learn obscure vendor-dependent test syntax but rather can define their own commands if desired. LINC collects alarms in real-time and displays customized network status.

The TIMS (Total Information Management System) is a 'smart' trouble tracking and inventory maintenance system which will allow quick, direct access to inventory information, component association, topology management, historical data, autogenning, trouble ticketing and tracking, and performance/availability reporting.

NETREX is the 'brain' that ties these subsystems into ISM. NETREX uses LINC and TIMS as its interfaces into the outside world. Having such subsystems already in place enables NETREX to be orders of magnitude more powerful than other expert systems.

The basic operational paradigm we propose for all NETREX modules is that of being problem driven. Thus a module is presented with a problem from a subsystem. For example, a new trouble ticket is generated by the trouble ticket subsystem, or a ticket on the pending stack is presented by the escalator subsystem, or the alarms event generator presents a significant problem event. The ES will then operate on the problem stack in the order of assigned priorities. The ES must be able to keep up with real-time events being generated by these problem generators. This could be as frequent as every minute in the case of alarms events. The ES will normally collect appropriate information from the relevant databases, form some initial *hypotheses* and then begin to *test* those hypotheses by issuing network sensory commands. The objective of the diagnosis is to isolate a fault to a correctable component, and to fix the fault via software network commands or ultimately automatically dispatch a fixit agent. If the system cannot solve the problem it must annotate its findings on an existing trouble ticket, or create a new trouble ticket and bring this to the attention of operators on a severity basis.

NETREX acts as another (high-speed) user of LINC and TIMS with special interfaces. Thus the boundaries between these subsystems are fairly distinct. However, there is a need for a tighter coupling with the alarms system being run by LINC. This is due to the sheer volume of alarms being generated, and the need for alarm correlations over several days. LINC has the capability of collecting raw alarms, filtering these alarms according to user-defined criteria, and presenting user-defined displays. For example, trouble center operators usually display only high-level alarms such as node and internodal link, so that they are not overwhelmed with information. NETREX also needs to receive alarms on a higher level. Thus a special process within LINC performs smart filtering of alarms and collection of these alarms into significant alarm events. These events may be simple raw alarms in the case of, say, a large node going down, or subtle problems that have occurred over a period of time (e.g. this line module has dropped and restarted three times in the last two days). The alarms problem generator then passes over as much information as possible on the condition. This is essential to cut down the volume of intercommunication between LINC and NETREX.

7. THE NETREX PROTOTYPE DESIGN CRITERIA AND GOALS

We have implemented and tested a prototype module of NETREX. This module is the trouble ticket expert outlined previously. The objective of implementing the prototype was to test the feasibility of the NETREX plan, and to demonstrate various proofs of principle, in particular to develop the real-time aspects of the system such as the querying of remote databases and the development of suitable interfaces. In this section we describe the various design criteria that were adopted to achieve a working prototype in one year.

The first major constraint was that the system would be developed at Caltech in Pasadena, and would thus have remotely to access TAC facilities in San Diego. Even though the final system will be integrated into TAC hosts, we felt it necessary to demonstrate that the brain of such a system could be remote from its sensors and manipulators into the network cloud. This constraint allowed us flexibility in determining the implementation platform. We chose to implement the system on a UNIX SUN 3/160 workstation. This decision allowed us access to sophisticated programming tools in C, and easy upward migration for more compute power. Much of the programming effort in a real-time system is in the interfaces, and C and UNIX provide a powerful environment for programming these interfaces.

The next major choice was the choice of an expert system shell. We chose the Teknowledge S1 shell, for a variety of reasons. First, it is written in C and has interfaces into C and UNIX. These are essential for the development of a real-time system. Second, S1 was available for the Tandem Computers used in TAC, and we wanted to keep open the possibility of porting to these machines in future. Third, the S1 syntax has a very readable Pascal style. This is important for transferring the technology over to P*B programmers. Although S1 did not have all our requirements for a suitable shell (non-monotonicity is a problem, and the representation scheme is limited), these other factors far outweighed the limitations. Also, because of the easy interface to C, many of these limitations could be bypassed by C programming. Finally, a new upward-compatible improved version of the product called Copernicus would be released by the time a production prototype would need to be fielded.

A major choice of operational paradigm was that the ES would emulate an expert human user in the sense that standard interfaces to P*B applications would be used. Thus the ES would interact with the applications via terminal emulation. This relieved P*B from having to provide any special interfaces into their applications. This was a major benefit in development time, although the resulting interfaces were not as fast as they would have been, application to application.

Having decided on the above constraints, we were in a position to define the operation of the trouble ticket expert module as follows. The ES would monitor the CCTAC database and look for open (active) trouble tickets. The ES would then try to diagnose the problem. The final result would initially be an annotation of the trouble ticket with the ES's findings, leading on to pro-active repair of the fault (if possible). In this way the findings and performance of the ES could first be assessed by comparison with closed out tickets. In order to achieve this performance the ES would need to interact with the trouble ticket database and the alarms database, and to issue active commands to probe the network.

We then narrowed the domain of the problem to deal with only one network—the BANCS bysync network. There were several reasons for this decision. First, the BANCS network is one of the largest within P*B and is well established with many CCTAC experts available. Secondly, the new LINC and TIMS applications were still undergoing modification and testing. Thus, although they could be accessed via the BANCS network, we would not have to rely on them being stable initially. The final prototype that we have implemented is thus a BANCS expert module which can interrogate the NDBS database for inventory and trouble ticket information, access alarms information, and issue active THP commands to obtain current sense and status information. Figure 2 shows a block diagram of how the system interacts with the network.

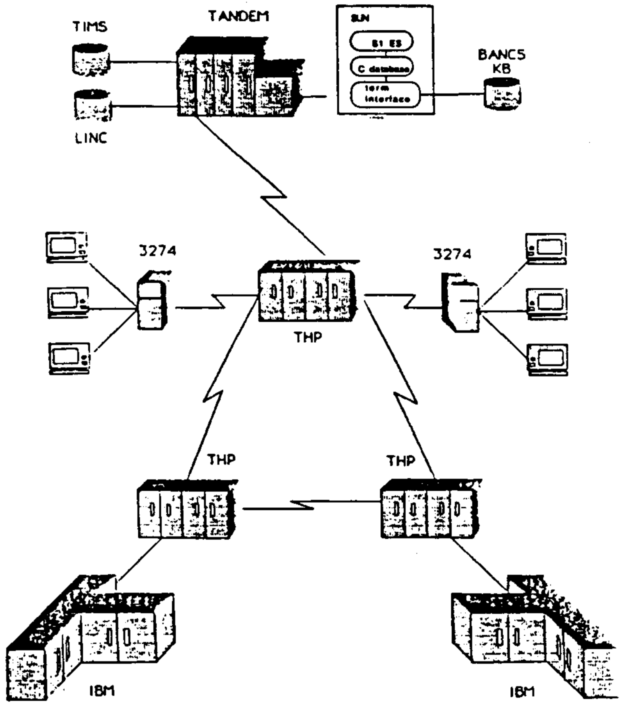

Fig. 2. BANCS prototype.

8. THE NETREX PROTOTYPE—ARCHITECTURE AND OPERATION

The ES acts as a regular terminal user. The facilities at the Caltech (development) end therefore consisted of an IBM 3274 control unit with a standard 3272 terminal, as would be used by a CCTAC data specialist. Also coming off the CU is a printer and the connection to the SUN. The SUN runs a commercially available 3272 emulation program, which handles the low-level terminal protocols and enables several virtual terminal sessions to be run simultaneously. The structure of the SUN software is shown in Figure 2. At the highest level the SUN is running UNIX, and the S1 shell which runs the prototype ES runs without that. When the ES needs external information a call is made to a data cache program written in C and running as an autonomous process within UNIX. The data cache program handles both the dialogue with S1 and the terminal emulator. In addition, information is buffered in the cache so that unnecessary requests to P*B databases are not made. This is essential given the low-speed link to P*B facilities in this prototype. However, any ES needs some form of externally programmed dialogue manager and local database if it is to operate in real-time. It is far too inefficient to attempt this process within the shell itself.

The cache system is composed of a database program, a set of remote-command functions, and a set of filter programs written for the UNIX *awk* tool. The database stores an arbitrary number of datasets. Each dataset can have any number of hash tables. These tables index the dataset by different keys. The remote-command functions are executed by the database program whenever a query would result in a 'not found' being returned. Instead of returning not found, the command-function formats a command or sequence of commands to the remote mainframe. The results of this query to the mainframe are passed through the *awk* filter and loaded into the database. The original query is attempted again. This time the results are returned directly even if the query result is 'not found.'

The cache system serves two purposes. The first one is to reduce the number of queries to the remote system. The queries from the expert system to the cache system are for one field in a large page of data. Since the expert system is likely to request other fields from the same page, the cache system usually does not have to execute a remote command to obtain the required data. The second purpose of the cache system is to insulate the expert system from changes in the interface at the remote end. The expert system does not have to be concerned with how the information is obtained. Changes in the screen format of output data from the remote mainframe require a change in the small *awk* filter program for that dataset. Changes in the syntax for queries to the remote mainframe require a change in the appropriate remote-command function. The entire cache system can be extended by adding the required remote-command function, writing an *awk* filter if it is needed, and changing the database table which describes the names and locations of database sets and database fields.

9. THE NETREX PROTOTYPE S1 PROGRAM

Much of the initial work in developing the NETREX prototype was in the interfaces, as opposed to intensive knowledge engineering. The rule-based knowledge came from both Comtech experts and equipment manuals. We felt that it was most important to get the structure of the expert system and its interfaces correct first, and then to develop the knowledge base later.

NETREX has been implemented using the S1/Copernicus expert system shell from Teknowledge. The knowledge representation scheme employed in this shell is based on object-attribute-value triples. Essentially, objects are specific items which the expert reasons about when solving problems; attributes are characteristics of these objects; and values are possible values of the attributes. The basic paradigm of operation is the determination of attribute values for specific instances of objects, via backward rule-chaining. Objects may range from specific physical devices (e.g. a printer or a multiplexer) to abstract notions such as 'problem' or 'intermediate symptoms'. Figure 3 shows some of the S1 NETREX rules.

In the NETREX trouble ticket prototype we perform diagnosis of 'network problems'. NETREX monitors the local cache database for new 'open' (unresolved) trouble tickets. The local cache database significantly simplifies the problem of interfacing the expert system shell to the real 'live' network, effectively buffering the two. Once a new ticket is detected, NETREX creates an instance of the ticket and then determines many of the attributes of this ticket in a procedural manner. For example, network ID data is extracted and a search is made for the most recent trouble ticket relating to this device. The ES also parses various parts of the free-form notes section of the ticket to try to assess the status of the human problem resolution on this ticket. For example, has it been referred to another agency? It is a feature of this shell that procedural code can be incorporated into the general inference scheme. In this case we found that the experts invariably 'eyeball' some initial information about the problem before considering any hypotheses.

The device on which the ticket is opened is then checked via a series of THP status commands both to verify the information on the ticket and to determine the status of surrounding and hierarchically connected components. As an example the DSS (display station status) command returns a list of abnormal conditions (line error codes, poll fail messages) for both the suspected device and related devices. The time of failure (as reported on the ticket) is also used to search the alarm log for any prior alarms for the device. This simple procedure is easy to perform in an automated system but is rarely used by the experts because of interface difficulties. At this point NETREX begins to diagnose the problem by backward chaining on three different attributes of the instance problem, namely, the level, nature and current status of the problem. Additional network commands may be invoked during this backward chaining if more data is required. Based on these intermediate attributes NETREX finally backward chains to determine the attributes actual.problem and recommended.solution. The result of the problem resolution is logged on a local copy of the ticket for future analysis. Figure 4 shows an example of the output of the NETREX prototype program.

10. CONCLUSIONS

In this paper we have presented a methodology for using real-time autonomous expert systems in network management. Our approach was intimately linked with the ISM system being implemented by Pacific Bell, but many aspects are generic.

```
DEFINE RULE                        DSS002.1
:: APPLIED.TO                      p : prob
:: PREMISE                         already.existing (t : terminal|last.terminal (t) and
                                   cfl.count [t] > 3)
:: CONCLUSION                      nature [p] = "related to other devices on the same
                                   control unit" <0.6>
END.DEFINE
/* Internal Slots for dss002
:: CONCLUDED.ATTRIBUTES (nature)
*/
DEFINE RULE                        DSS002.2
::APPLIED.TO                       p : prob
:: PREMISE                         already.existing (t : terminal|last.terminal (t) and
                                   cfl.count [t] > 3)
::CONCLUSION                       level [p] = "line" <0.5>, "control.unit" <0.6>
END.DEFINE
/* Internal Slots for dss002
:: CONCLUDED.ATTRIBUTES (nature)
*/
DEFINE RULE                        DSS003
:: APPLIED.TO                      p : prob
:: PREMISE                         already.existing (t : terminal|last.terminal (t) and
                                   dss.modifier [t] ~ = "no polling problems")
:: CONCLUSION                      current.status [p] = "there is a polling problem"
END.DEFINE
/* Internal Slots for dss003
:: CONCLUDED.ATTRIBUTES (current.status)
*/
DEFINE RULE                        EBC4050.1
:: APPLIED.TO                      p : prob
:: PREMISE                         already.existing (t : terminal|last.terminal (t) and
                                   match.strings (alarml.string [t], "EBCDIC
                                   STATUS *4050*"))
:: CONCLUSION                      current.status [p] = "the device does not exist
                                   from the CU's viewpoint"
END.DEFINE
/* Internal Slots for ebc4050.1
:: CONCLUDED.ATTRIBUTES (current.status)
*/
DEFINE RULE                        EBC4050.2
:: APPLIED.TO                      p : prob
:: PREMISE                         already.existing (t : terminal|last.terminal (t) and
                                   match.strings (alarm1.string [t], "EBCDIC
                                   STATUS *4050*"))
:: CONCLUSION                      level [p] = "station"
END.DEFINE
/* Internal Slots for ebc4050.2
:: CONCLUDED.ATTRIBUTES (level)
*/
```

Fig. 3.

The NETREX BANCS prototype has satisfactorily demonstrated several important proofs of principle, validating our design methodologies, the most important of which was the real-time accessing of external databases and the issuing of active network commands. The 'expertise' shown by the prototype was secondary. However, it has emerged that data collection, digestion and presentation are well suited to ES

```
We are looking at a new trouble
This trouble ticket will be referred to as ticket 6
The time currently is 1412
Ticket data obtained from NDBS:
   ticket 6  field:trouble ticket number           value:304314
   ticket 6  field:device                          value:F026207
   ticket 6  field:component type                  value:STAT
   ticket 6  field:device type                     value:4540PT
   ticket 6  field:date originated                 value:08 16 88
   ticket 6  field:time originated                 value: 07 54
   ticket 6  field:ticket referred to              value:OCS
   ticket 6  field:previous ticket no.             value:
  This ticket has been open for 1 days and 6 hours and 18 minutes
  The most recent ticket for this device is not stored in the NDBS database
   THP report data on terminal  #4:
   terminal  #4  field:network address             value:F026207
   terminal  #4  field:terminal type               value:4TP
   terminal  #4  field:application                 value:D58
   terminal  #4  field:log.status                  value:ON
   terminal  #4  field:inactive                    value:
   terminal  #4  field:autolog                     value: X
   terminal  #4  field:dss result                  value:no polling problems
   Alarms data from NDBS:
   terminal  #4  field:alarm message               value:CAL FAIL NAK
   terminal  #4 field:alarm message                value:EBCDIC STATUS *4050*
Analysis of the data:
  The problem appears to be at the station <0.9> level
The nature of the problem appears to be: transmission errors <1.0>, and severe <0.3>
The current status is that there are no polling problems with the device <1.0>, the device is logged on
<1.0>, the device is CFL due to negative acknowledgments to a THP message <1.0>, and degraded
service <0.6>
```

Fig. 4.

technology. Also, the prototype was capable of useful if low-level diagnosis and repair (say recycling a line to bring it up again). The prototype at the time of writing is still being assessed, and its knowledge base significantly increased. Initial performance of this simple version of NETREX has been quite encouraging and with the addition of more sophisticated techniques such as the correlation of multiple problems due to one source it is expected to be deployed in a production version within the next 12 months. NETREX prototype 2 is concurrently under development and will incorporate even more powerful features. Firstly, the C interfaces to the TIMS database and LINC control system are being developed, and these will function application to application via X.25 links. This will open up high-speed access to more networks and components and enable a more sophisticated network model to be built up by the ES. Secondly, the system will become alarm driven. Also, prototype 2 is being written in Copernicus, the updated shell. This will allow more efficient dialogue with the C and UNIX environment, and provide much-needed ES features such as non-monotonicity. Overall, given our experience in developing the prototype, we feel that the design goals for the fully integrated NETREX system outlined earlier are quite feasible with current ES technology.

ACKNOWLEDGMENT

This work is funded in part by a grant from Pacific Bell.

REFERENCES

1 Knowledge based systems for communications. IEEE Journal on Selected Areas in Communications, 6(5) June (1988).
2 Expert systems in network management. IEEE Network Magazine, 2(5) (1988).
3 C.E. Clark, A knowledge-based information display tool for network planning. Workshop on the Integration of Expert Systems into Network Operations, IEEE ICC '87, Seattle, Washington, June 1987.
4 I.A. Ferguson and D.R. Zlatin, Generic strategies and representations for communications networks design and sales. IEEE Network Magazine, 2(5) (1988).
5 S. Bernstein, DESIGNet: an intelligent system for network design and modelling. Workshop on the Integration of Expert Systems into Network Operations, IEEE ICC '87, Seattle, Washington, June 1987.
6 W.D. Zhan, S. Thanawastien and L.M.L. Delacambre, SIMNETMAN: an expert workstation for designing rule-based network management systems. Proc. IEEE 1988 Network Operations and Management Symposium, New Orleans, Louisiana, February 1988.
7 F.M. Miller, G.V.E Otto, E.M. Siegfried and P.E. Zeldin, ACE: a knowledge based maintenance analyzer. Proc. IEEE Automated Testing Conference, 1985.
8 T.L. Williams, P.J. Orgren and C.L. Smith, Diagnosis of multiple faults in a nationwide communications network. Proc. 8th IJICAI, Karlsruhe, West Germany, August 1983.
9 T.E. Marques, A symptom driven expert system for isolating and correcting network faults. IEEE Communications Magazine, 26(3) (1988).
10 N. Kahn, P. Callahan, R. Dube, J. Tsay and W. Van Dusen, An engineering approach to model based trouble shooting in communications networks. Proc. IEEE GLOBECOM '87, Tokyo, Japan, November 1987.
11 S. Guattery and F.J. Villarreal, NEMYSYS: an expert system for fighting congestion in the long distance network. Proc. Expert Systems in Government Symposium, K. Karna (ed.), McLean, Virginia October 1985.
12 S. Rabie, A. Rau-Chaplin and T. Shibahara, DAD: a real time expert system for monitoring of data packet networks. IEEE Network Magazine, 2(5) 1988.
13 S.K. Goyal, D.S. Prerau, A.V. Lemmon, A.S. Gunderson and R.E. Reinke, Compass: an expert system for telephone switch maintenance. Proc. Expert Systems in Government Symposium, K. Karna (ed.), McLean, Virginia, October 1985.
14 T. Bult, V. Peacocke, S. Rabie and V.S. Carter, DMS-100 maintenance advisor: on-line expertise. TELESIS (BNR) 4 (1986).
15 M. Sutter, Smart questions, smart answers. Telephony, June 2 (1986).
16 J.H. Griesmer, YES/MVS: a continuous real time expert system. Proc. AAAI-84, Austin, Texas, 1984.
17 W.R. Nelson, REACTOR: an expert system for diagnosis and treatment of nuclear reactor accidents. Proc. AAAI-82, 296–301, 1982.
18 L.M. Fagan, J.C. Kunz, E.A. Feigenbaum and J.J. Osborn. Representation of dynamic clinical knowledge: measurement and interpretation in the ICU. Proc. IJCAI-79, pp. 260–262, 1979.

19 R. Mathonet, H. Van Cotthem and L. Vanryckeghem. Dantes: an expert system for real time network troubleshooting. Proc. IJCAI '87, Milan, Italy, August 1987.
20 M.T. Sutter and P.E. Zeldin, Designing expert systems for real time diagnosis of self correcting networks. IEEE Network Magazine, 2(5) (1988).
21 R.M.F. Goodman and P. Smyth, Decision tree design from a communication theory standpoint. IEEE Transactions on Information Theory, 1987 (in press).
22 R.M.F. Goodman and P. Smyth, Learning from examples using information theory. Proceedings of the Second Workshop on Knowledge Acquisition, Banff, Canada, October 1987.
23 T. Chiueh and R.M.F. Goodman. A neural network classifier based on coding theory. IEEE Conference on Neural Information Processing Systems-Natural and Synthetic, Denver, Colorado, November 1987.
24 R.M.F. Goodman and P. Smyth, An information theoretic model for rule-based expert systems. Presented at the 1988 IEEE International Symposium on Information Theory, Kobe, Japan, 1988.
25 R.M.F. Goodman and P. Smyth, ITRULE: an information-theoretic rule induction algorithm. Presented at the First European Workshop on Knowledge Acquisition, Reading, England, 1987.
26 R.M.F. Goodman and P. Smyth, Information theory expert systems, and neural networks. 1988 Beijing International Workshop on Information Theory, Beijing, China, July 1988.
27 R.M.F. Goodman and P. Smyth, Information-theoretic rule induction. 1988 European Conference on Artificial Intelligence. Munich, Germany, 2–5 August 1988.
28 T. Chiueh and R.M.F. Goodman, High capacity exponential associative memories. IEEE Annual International Conference on Neural Networks, San Diego, California, 24–27 July, 1988.
29 T. Chiueh and R.M.F. Goodman, Learning algorithms for neural networks with ternary weights. 1988 Annual Meeting of the International Neural Network Society, Boston, Massachusetts, September 1988.
30 R.M.F. Goodman, J.W. Miller and P. Smyth, An information theoretic approach to rule-based connectionist expert systems. Presented at IEEE Conference on Neural Information Processing Systems-Natural and Synthetic, Denver, Colorado, November 1988.
31 T. Chiueh and R.M.F. Goodman, Controlling classification regions via backpropagation. IEEE Conference on Neural Information Processing Systems-Natural and Synthetic, Denver, Colorado, November 1988 (submitted).
32 R.M.F. Goodman and P. Smyth, On rule-based probabilistic inference: theoretical principles and practical techniques. Presented at AIII Conference on AI and Statistics, 1988.
33 R.M.F. Goodman and P. Smyth, Automated rule acquisition. Third AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, 6–11 November 1988.
34 R.M.F. Goodman, H. Latin, J.W. Miller and P. Smyth, NetRex: a real time network management expert system. IEEE Globecom '88 Workshop on the Application of Emerging Technologies in Network Operations and Management, December 1988 (submitted).