Event Correlation using Rule and Object Based Techniques

Y. A. Nygate
AT&T - Bell Laboratories
6200 E. Broad St. - Rm. 2B253, Columbus, OH 43213, U.S.A.
Tel: 614-860-5976 Fax: 614-868-4021 email: yossi@hercules.cb.att.com

Abstract
Today's competitive market place has forced the telecommunications industry to improve their service and reliability. One step that telecommunications companies have taken to reduce network failures is the installation of operations centers to collect data from network elements. These centers are staffed by network managers who monitor network activity by correlating alarms across various operational disciplines (switch, facility, traffic) and relating them to a common cause. Accurate analysis is often difficult due to the volume of data and complexity of problems.

ECXpert is a product developed recently at AT&T to help network managers monitor and analyze alarms, take corrective actions, and minimize disruptions to the network. Successful implementation of event correlation has increased customer revenue since trouble isolation can be done faster, resulting in quicker restoration of service.

The essence of ECXpert is a high level language with which users can specify network events and their correlation with alarms. The system is written in Prolog and C++, a powerful combination which facilitated development to occur on time and in budget. It has been deployed in network management centers throughout the U.S. and is currently being marketed overseas. ECXpert is a success story for Prolog within AT&T.

Keywords
Network Management, TNM, Event Correlation, C++, Prolog, Meta-Languages

1 INTRODUCTION

Total Network Management (Nerys, 1993), TNM, is a very large product developed by AT&T for domestic and international customers. The primary function of TNM is to facilitate early problem detection and prompt repair of telecommunication network facilities and switches. TNM users monitor line-oriented displays to analyze alarms generated by failures, correlate the alarms with knowledge of problem scenarios, and build up a picture of network events. If necessary, a repair request is generated and users continue monitoring to verify that either the problem cleared up or that further action is needed to solve the problem.

Timely generation of repair requests in response to a large volume of alarms has been notoriously difficult to do well. One minor problem, that by itself may be of little importance, can create a major problem when combined with other minor problems. In contrast, one major problem might generate many additional minor problems. Typically, there are many problems occurring concurrently in the network resulting in hundreds of active alarms intermingled on the displays. Users need to group the alarms corresponding to problems, differentiate between alarms that are the underlying causes and those that are results, generate repair orders, and monitor resolution of the problem. Due to the volume of data and complexity of modern telecommunication networks, this task has been difficult to do successfully. This problem is often called *event correlation* and can be defined as the analysis and classification of multiple messages from one or more sources to determine the underlying cause of a failure. The results of correlating alarms correctly can be used to relate the resultant impact and symptomatic troubles to the underlying causes.

Successful implementation of event correlation has increased customers' revenue because trouble isolation can be done faster, resulting in quicker restoration of service. Relating cause to service impact allows prioritization of repairs so problems that cause service outage and loss of revenue can be assigned high priority.

This paper describes the Event Correlation Expert feature package, ECXpert, that was incorporated into the TNM product family to help network managers speedily resolve the problems of analysis, recognition and resolution of alarms.

2 EVENT CORRELATION IN TELECOMMUNICATIONS

2.1 Monitoring Alarms in Telephone Networks

TNM's primary function is to collect, process, and display messages received from network elements (NEs). A typical TNM center may collect thousands of alarms per hour and depending on the type of message, a minor, major or critical alarm may be generated or a previously received alarm may be cleared. Network managers monitor the alarms using line-oriented displays known as awareness screens (AS) that update every 6 seconds. The user can see up to 16 alarms on each page of the AS and can page forward and backward to see other alarms. Each alarm is displayed in a color corresponding to its severity: red for the most severe, blue for the least severe, and green for cleared alarms. The main responsibility of the network managers is to monitor the alarms, picture the current underlying network problems, generate the necessary repair orders, and continue monitoring the network to verify that their analysis was correct and the problem was resolved.

To explain event correlation, I shall present a running example in which the network includes an AT&T 5ESS® switch connected to two other switches made by AT&T and another manufacturer. Between each NE there are a pair of links that together comprise a path. In our example, three network events occur. At 4:24 a hardware failure (X1) occurs on the link between *clli_a* and *clli_y*, at 4:50 high traffic demand (X2) occurs at *clli_z*, and at 5:22 another hardware failure (X3) occurs on the second link between *clli_a* and *clli_y*. The network and location of the events are shown in Figure 1.

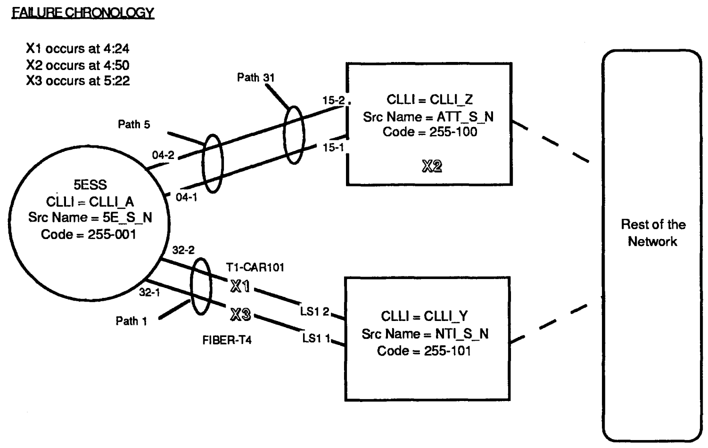

Figure 1 Example Network and Failures.

In telecommunication networks, there are often many more alarms than number of failures as a combination of failures can create additional problems which may result in other alarms being generated. For example, after X1 and X3 occur since all the links between *clli_a* and *clli_y* fail a *path loss* message will be generated. After all three failures occur, since it is impossible for traffic to leave *clli_a* (as *clli_z* and the two links to *clli_y* have failed) a *switch isolation* message will be generated. The set of generated alarms is a function of the nature of the problems, their location and time of occurrence, as well as the configuration of the network. In our example, 19 alarms were generated and displayed on the AS as shown in Figure 2. If the same three events had occurred in different places in the network, possibly only three alarms might have been generated.

DATE	TIME	SYSTEM	A OFFICE	Z OFFICE	TROUBLE INDICATION
09jun	5:30	nti_s_n	CLLI_Y	CLLI_A	LINK FAIL LS1 1
09jun	5:28	5e_s_n	CLLI_A	CLLI_Y	SWITCH ISO 255-001
09jun	5:26	5e_s_n	CLLI_A	CLLI_Y	PATH LOSS 1
09jun	5:24	5e_s_n	CLLI_A	CLLI_Y	LINK FAIL 32-1
09jun	5:22	FIBER-T4	CLLI_E	CLLI_R	HARDWARE FAIL
09jun	5:06	att_s_n	CLLI_Z	CLLI_A	PATH LOSS 31
09jun	5:04	att_s_n	CLLI_Z	CLLI_A	LINK FAIL 15-1
09jun	5:02	att_s_n	CLLI_Z	CLLI_A	LINK FAIL 15-2
09jun	5:00	5e_s_n	CLLI_A	CLLI_Z	PATH LOSS 5
09jun	4:58	5e_s_n	CLLI_A	CLLI_Z	LINK FAIL 04-1
09jun	4:56	5e_s_n	CLLI_A	CLLI_Z	LINK FAIL 04-2
09jun	4:56	att_s_n	CLLI_Z	CLLI_A	CNGSTN RESTART 15-1
09jun	4:55	5e_s_n	CLLI_A	CLLI_Z	OVERLOAD FAIL 04-1
09jun	4:54	5e_s_n	CLLI_A	CLLI_Z	OVERLOAD FAIL 04-2
09jun	4:52	att_s_n	CLLI_Z	CLLI_A	CNGSTN RESTART 15-2
09jun	4:50	att_s_n	CLLI_Z		HI TRAFFIC DMND
09jun	4:29	nti_s_n	CLLI_Y	CLLI_A	LINK FAIL LS1 2
09jun	4:27	5e_s_n	CLLI_A	CLLI_Y	LINK FAIL 32-2
09jun	4:24	T1-CAR101	CLLI_D	CLLI_Q	HARDWARE FAIL

Figure 2 Alarms Generated by Network Failures.

In large networks, many problems occur concurrently resulting in thousands of active alarms. Network managers would be overwhelmed if all these alarms were displayed on every users awareness screen. TNM allows users to specify viewing options that restrict which alarms are displayed (such as specific switch types, or regions) and sorting options (such as by time or severity) These options have to be used prudently. When their view is too restrictive it is often difficult to see the 'big picture' of network problems. Whereas if they do not make enough restrictions, they are unable to read the alarms fast enough to keep up with the flow of information across their screens.

Although useful, viewing options do not utilize the underlying cause and effect relationship that exists between events and alarms. Consequently, an AS will often contain many pages of alarms caused by different problems intermingled on the screen. Because these alarms are not grouped together, it is very difficult to construct an accurate picture of the network problems and differentiate between alarms that are the underlying causes and those that are results.

2.2 Correlation Trees and Correlation Groups

Many network problems can be depicted as a combination of alarms having a specific cause and effect relationship. This can be depicted schematically in a *correlation tree skeleton* as shown in Figure 3.

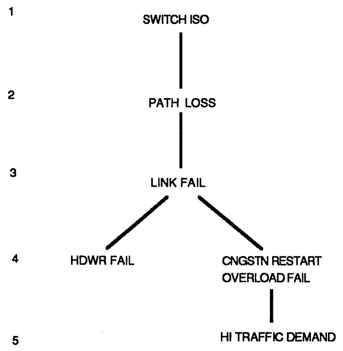

Figure 3 Correlation Tree Skeleton.

```
cor_group = 1 time window = 60 minutes

if new_msg.Trouble[1-2] = path loss precedence = 2 or
if new_msg.Trouble[1-2]=overload fail precedence =4 or
if new_msg.Trouble[1-2]=cngstn restart precedence =4
    new_msg correlates old_msg when
    case old_msg.Trouble[1-3] = hi traffic dmnd
        new_msg.A_Office = old_msg.A_Office or
        new_msg.Z_Office = old_msg.A_Office
    case old_msg.Trouble = anything_else
        (new_msg.A_Office =old_msg.A_Office and
        new_msg.Z_Office = old_msg.Z_Office) or
        (new_msg.A_Office = old_msg.Z_Office and
        new_msg.Z_Office = old_msg.A_Office)
```

Figure 4 Correlation Group.

In these trees, the child/parent link is equivalent to a cause and effect relationship between messages. Equivalent messages (such as *cngstn restart* or *overload fail*) are on the same node. Alternative children to a parent are similar to 'or ' branches, that is a *link fail* can cause a *path loss* whereas either a *hardware fail* or *cngstn restart /overload fail* can cause a *link fail.* However, it does not imply that every *link fail* will always cause a *path loss.* For example, one needs all the links between NEs to fail before a path loss occurs. This representation was chosen because users found it intuitive as they often discussed problems in terms of cause and effect. Using these skeletons, the 19 alarms that were generated by the three failures can be represented as a *correlation tree instances* as shown in Figure 5 below.

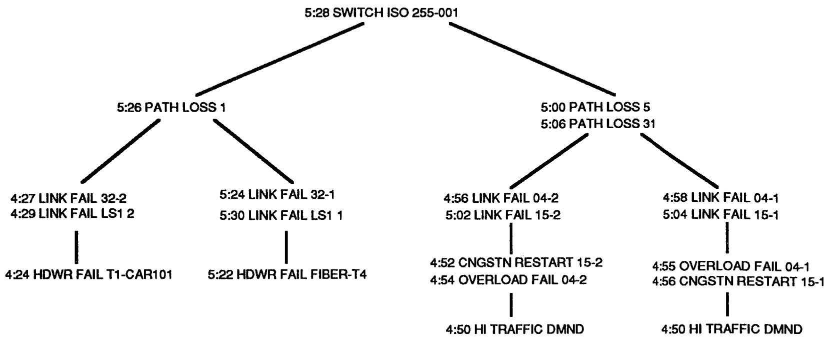

Figure 5 Correlation Tree Instance.

Each node in the correlation tree is a group of one or more equivalent messages (e.g. the *overload fail* at 4:55 and the *cngstn restart* at 4:56) and two nodes are connected if there is a cause and effect relationship between them (e.g. the *link fail* at 4:58 was one of the causes of the *path loss* at 5:00). Each branch of the correlation tree corresponds to a branch in the correlation tree skeleton in Figure 3 with the leaves being the underlying causes of a current network problem and the root being the result. In our example the skeleton contains a *path loss* causing a *switch iso*. Since both *path loss*es correlate the *switch iso* and did not correlate each other they became separate children of the *switch iso*. In our example the 4 leaves (two *hdwr fail*s and two *hi traffic dmnd*s are ultimately the cause of the *switch iso*.

3 ECXPERT

3.1 Correlation Grammar

The primary role of ECXpert is to receive alarms and to dynamically create correlation trees based on the correlation tree skeletons. Since TNM is sold to many customers - each having different levels of network management expertise, performance and security constraints - this package needs to configurable in the field by the customer. ECXpert supports an expert system shell that provides a pseudo-English description language in which users define *correlation groups* that correspond to the correlation tree skeletons. Each correlation group can be viewed as a model of a particular network problem. Using this language users specify

- when a new alarm belongs to a correlation group;
- when a new alarm correlates previous alarms that belonged to this group;
- the cause and effect relationship between alarms;
- what actions to take - e.g. automatic generation of a trouble ticket or invoking a reroute;
- a time window - that is only correlate alarms that have occurred within this time window.

The correlation grammar also allows rules to execute database look ups. Administrators can, then, write correlation groups to make use of network configuration data. For example, a rule might correlate the *linkfails* occurring at 4:27 and 4:29 only if they are physically the same link. Figure 4 above, shows some of the rules used to correlate the alarms shown in the correlation tree skeleton in Figure 3.

A correlation group is comprised of three parts. The first part specifies the correlation group number used when displaying the correlation trees and a time window. The second part assigns a precedence to each type of message in this group which corresponds to the level in the correlation tree skeleton. The third part defines when a new message correlates an old(er) message in the group. For example, this rule states that if the first two words in the trouble field of a newly received message are *path loss*, *cngstn restart*, or *overload fail*, the message will belong to correlation group 1 with precedences 2, 4, and 4 respectively. Furthermore, they will correlate an older message in this group whose first three words in the trouble field are *hi traffic dmnd* if the new message's *a_office* field or *z_office* field is the same as the old message's *a_office* field and both alarms occurred less than 60 minutes apart. The grammar allows administrator to define macros; use arithmetic and string comparisons including the use of regular expression; and specify correlation conditions using logical 'and's, 'or's, and parentheses.

In a correlation group of *n* types of messages, there are n^2 possible correlation rules. This could be very large and cumbersome to write and maintain. To reduce the amount of typing, the correlation grammar provides constructs to allow many messages of the same type to be grouped together. For example, if we look at the rules in Figure 4, the *path loss*, *cngstn restart*, and *overload fail* all use the same correlation rules. In addition, there is one specific correlation condition for the *hi traffic dmnd* message, all the other messages in the group are correlated using the *anything_else* (default) correlation rule clause. This is both intuitive to the users and compact. Using this shorthand notation, most correlation groups have O(nlog(n)) lines with respect to the number of message types.

3.2 Dynamic Manipulation of Correlation Trees.

During normal operation, TNM receives messages from NEs and checks whether the message is

- An alarm - indicating a new problem and is then assigned a severity level and is displayed on the AS and sent to ECXpert for processing.
- A clear message - indicating a previous problem that caused an alarm has been corrected and can be removed from the AS and sent to ECXpert for processing.
- An informational message that can be ignored.

As each alarm is received, ECXpert uses the correlation conditions defined by the correlation groups to add the new alarm to all the relevant correlation trees. The algorithm to do this is quite complex and beyond the scope of this paper. A complete description of the algorithm can be found in (Nygate, 1994). In general, as each alarm is processed one or more of the following actions are taken. The new alarm may

- Be added to a tree - indicating that this alarm is part of a larger problem.
- Start a new tree - indicating a new problem.
- Combine a number of trees - indicating that what were a few small problems before were really only part of a larger one.
- Split a tree - indicating that an underlying cause is responsible for two or more problems.

- Clear an old message - indicating that this problem has now been resolved. This causes the tree to begin to decompose and if nothing new is added; it will eventually disappear.

4 HOW ECXPERT WAS DEVELOPED

The use of single knowledge representations and techniques for knowledge based system has been widely used (Abelson and Sussman, 1985). Successful applications have been reported in the literature in diagnostic systems (Shortliffe, 1976), planners (Ambros-Ingerson and Steel, 1988) and heuristic classification (Clancey, 1983).

However, many problems do not suit the problem solving characteristics of any one particular technique and need to be attacked by a variety of methods. Advocates of applying multiple methods in a single system (Fikes and Kehler 1985) contend that just as a carpenter has many tools, each specialized to its purpose, so should there be many tools in the programmer's kit (Bobrow and Stefik, 1986). Trying to solve a problem that does not fit well into a particular technique may result in programs that are buggy, slow, awkward and long. However, integrating multiple methods does incur a cost. For example, modules may be required to transform between different representations of the same information to optimize processing. But, if the cost is small, the benefits are great. Programmers can choose the most applicable problem solving technique to the module in question.

ECXpert integrates C++ and Prolog in a design that utilizes the run-time efficiency and support for object oriented design of C++ with the powerful meta-programming, semantic parsing, and pattern matching features of Prolog. The design and development of ECXpert was based on ASPEN (Nygate and Sterling, 1993), a new multi-paradigm method for developing knowledge based systems. ASPEN draws on the strengths of those that tout the clarity and success of single problem solving techniques with those who advocate the power and flexibility of multiple methods for software development. This compromise is achieved by providing a structured decomposition that allows each module to use different knowledge based techniques while defining a set number of modules with well delimited borders and functionalities. More information on ASPEN can be found in (Nygate, 1994).

ECXpert is comprised of four main modules - a correlation process, a correlation group compiler, a test correlation process, and a user interface.

4.1 Correlation Process

The correlation process is comprised of a C++ object for collecting alarms from TNM, a Prolog object for executing the correlation algorithm, and a C++ object to manipulate the database containing the composite alarm objects, that is the correlation trees.

The Prolog object can be viewed as a forward-chaining correlation engine that takes each alarm, find what rules it matches and fires a set of rules to update the correlation trees. As each alarm is processed, the Prolog object determines with which trees the new message correlates. Then using the algorithm mentioned in section 3.2, it determines the actions the database object must execute to update the correlation trees.

The correlation process contains 5000 line of C++ code, including 1500 lines for data structure conversion between C++ objects and Prolog lists; 1700 lines of static Prolog code to implement the correlation algorithm; and typically 2500 lines of user-supplied correlation rules.

4.2 The Correlation Group Compiler

The correlation group compiler is written in Prolog and converts user supplied correlation rules into Prolog Horn clauses (Kowalski, 1979). These rules are then dynamically linked with the correlation process. The syntax of the correlation grammar can be specified using a Definite Clause Grammar. Prolog's support for DCGs made the code generation straightforward. As mentioned in section 3.1, the correlation grammar provides a compact notation for combining multiple correlation rules by using 'or's. The compiler expands the disjunctions on the left hand side of the correlation rules and convert them into Horn Clauses. More information on the use of Prolog in ECXpert can be found in (Nygate, 1994).

The development of ECXpert is typical of the meta-programming approach to develop knowledge-based systems as advocated by (Sterling, 1990) and described by (Ümit Yalçinalp, 1991) in her Ph.D. thesis. A high-level language is developed for the application which can be easily compiled into Prolog or executed directly with a simple interpreter.

A typical correlation group contains 100 lines which is compiled into about 250 lines of Prolog code in 5 seconds. The tokenizer, code generator, and a user friendly error handling subsystem to help administrators find and fix syntax errors totals 1500 lines of Prolog code.

4.3 User Interface

When a user selects an alarm on the awareness screen, this module retrieves all the correlation trees to which this alarm belongs and displays them in a pop-up window on the AS monitors. For example, suppose the user selected the *link fail* received at 5:24 on the awareness screen, the correlation window that would be displayed is shown in Figure 6 below.

```
                         ********** CORRELATION WINDOW **********
PRECEDENCE DATE   TIME  L  SYSTEM      A OFFICE  Z OFFICE  TROUBLE INDICATION
SELECTED:  09jun  5:24  2  5e_s_n      CLLI_A    CLLI_Y    LINK FAIL 32-1
                  ***************** CORRELATION GROUP 1 *****************
1          09jun  5:28  2  5e_s_n      CLLI_A    CLLI_Y    SWITCH ISO 255-001
  2        09jun  5:26  2  5e_s_n      CLLI_A    CLLI_Y    PATH LOSS 1
    3      09jun  5:30  2  nti_s_n     CLLI_Y    CLLI_A    LINK FAIL LS1 1
    +      09jun  5:24  2  5e_s_n      CLLI_A    CLLI_Y    LINK FAIL 32-1
      4    09jun  5:22  1  FIBER-T4    CLLI_E    CLLI_R    HARDWARE FAIL
    3      09jun  4:29  2  nti_s_n     CLLI_Y    CLLI_A    LINK FAIL LS1 2
    +      09jun  4:27  2  5e_s_n      CLLI_A    CLLI_Y    LINK FAIL 32-2
      4    09jun  4:24  1  T1-CAR101   CLLI_D    CLLI_Q    HARDWARE FAIL
  2        09jun  5:06  2  att_s_n     CLLI_Z    CLLI_A    PATH LOSS 31
  +        09jun  5:00  2  5e_s_n      CLLI_A    CLLI_Z    PATH LOSS 5
    3      09jun  5:04  2  att_s_n     CLLI_Z    CLLI_A    LINK FAIL 15-1
    +      09jun  4:58  2  5e_s_n      CLLI_A    CLLI_Z    LINK FAIL 04-1
      4    09jun  4:56  2  5e_s_n      CLLI_A    CLLI_Z    CNGSTN RESTART 15-1
      +    09jun  4:55  2  att_s_n     CLLI_Z    CLLI_A    OVERLOAD FAIL 04-1
        5  09jun  4:50  2  att_s_n     CLLI_Z              HI TRAFFIC DMND
    3      09jun  5:02  2  att_s_n     CLLI_Z    CLLI_A    LINK FAIL 15-2
    +      09jun  4:56  2  5e_s_n      CLLI_A    CLLI_Z    LINK FAIL 04-2
      4    09jun  4:54  2  att_s_n     CLLI_A    CLLI_Z    OVERLOAD FAIL 04-2
      +    09jun  4:52  2  att_s_n     CLLI_Z    CLLI_A    CNGSTN RESTART 15-2
        5  09jun  4:50  2  att_s_n     CLLI_Z              HI TRAFFIC DMND
```

Figure 6 Correlation Window.

The precedence column corresponds to the precedence in the correlation grammar which allows users to reconstruct the correlation tree. The rest of the columns contain relevant data that also appeared on the awareness screen. For example, the *hi traffic dmnd* at 4:50 is a child of the *cngstn restart* at 4:52; and the *link fail* at 4:56 and the *link fail* at 4:58 are both children of the *path loss* at 5:00 which is in turn the child of the *switch iso* at 5:28. The *overload fail* at 4:55 and the *cngstn restart* at 4:56 are equivalent messages with the *cngstn restart* being the primary message.

Alarms in the correlation window are displayed in the same colors as on the AS, red for the most severe, blue for the least, and cleared alarms in green.

Since many groups can be active at once, the selected message can be in more than one group and each group can span more than one page. The user is able to scroll forward and backwards in the correlation screen looking at each group and page.

The correlation algorithm can also handle missing data. If, for example, neither of the *path loss* messages at 4:58 and 5:04 were received, the *overload fail* at 4:55 and *cngstn restart* at 4:56 would have become children of the *path loss* at 5:06. The precedence column of the correlation window would display a minus sign to signify that a message was missing as shown in Figure 7.

2	09jun	5:06	2	att_s_n	CLLI_Z	CLLI_A	PATH LOSS 31
+	09jun	5:00	2	5e_s_n	CLLI_A	CLLI_Z	PATH LOSS 5
-							
4	09jun	4:56	2	5e_s_n	CLLI_A	CLLI_Z	CNGSTN RESTART 15-1
+	09jun	4:55	2	att_s_n	CLLI_Z	CLLI_A	OVERLOAD FAIL 04-1
5	09jun	4:50	2	att_s_n	CLLI_Z		HI TRAFFIC DMND

Figure 7 Correlation Window with Missing Messages.

4.4 Test Correlation Process

To facilitate the administrator's role in writing correlation groups we provided a grammar that was intuitive, powerful and compact. In addition, once all the syntax errors in the correlation group were fixed, the administrators were able to verify that the semantics of the correlation group were correct using a test correlation process that incorporated the 'how' and 'why' (Sterling and Shapiro, 1986) tools used in Expert Systems. Administrators were able to provide an input file of high level alarms using the same format as displayed on the AS, send them one by one through a test correlation process and find out 'why' certain messages belonged to which correlation group and 'how ' they correlated with other, older, messages in the group. Once they were satisfied with the results, they could then install the correlation group.

5 USING THE CORRELATION TREE

ECXpert and its use of correlation trees provides many powerful ways of enhancing the effectiveness of network managers. The most obvious and direct improvement utilizes the fact that the leaves of the tree are the causes of the network problem with the root being the consequence. Thus in our example, if a user sees a *switch iso*, he/she can bring up the corresponding correlation window and see that the causes (leaves) are the two *hardware fail*s and the *hi traffic dmnd* and dispatch a repair order to fix these problems immediately. Each of the 'leaf' alarms occur frequently and they typically do not have any major network impact. Without correlation the leaf alarm would not have been fixed as quickly as other more obvious alarms. Once one of the leaves is fixed, all the messages in its branch often become cleared as well. If enough leaves are cleared, the root becomes cleared too. This is clearly shown in the correlation window and allows the user to execute retroactive analysis to see what combination of alarms (i.e. leaves) caused a network event, and how it was resolved (i.e. green branches).

A far more sophisticated but extremely useful feature of ECXpert, is to display on the AS only the alarms that correspond to the roots and the leave in the correlation tree while suppressing intermediate nodes in the tree. This has the immediate impact of reducing clutter on the awareness screens while leaving the critical nodes that show the overall network problems with their corresponding underlying causes.

Users can also set the AS restrictions to show a specific class or set of alarms. Whenever the Event Correlation window is invoked, all the alarms that correlate with the selected alarm are shown. This allows users to peruse the high level alarms but still have access on demand to all the low level contributing alarms. Other features include

- Escalating all the alarms in the correlation tree to the severity of the most severe alarm in that tree. For example, if a critical alarm (displayed as red) is added to a tree, all the alarms in the tree would be escalated and be displayed in red.
- Predicting what other problems must occur before a more serious network situation will occur. This is a very powerful feature as it allows users to estimate how far the network is away from a catastrophe and they can then protect/reserve the critical remaining resources.
- Allowing users to define actions in the correlation group such as setting off audible alarms (particularly useful during night shifts!), generating reports, generating new alarms, automatically starting a repair procedure, etc.

6 RESULTS AND EXTENSIONS

ECXpert has been installed in a number of sites in the U.S. and Europe. The initial customers, at NYNEX (NET and NYT), have been using event correlation to manage their SS7 network since 1992. Many other TNM users have since purchased ECXpert including PacBell, Bell South, SNET, and Bell Atlantic, to correlate alarms in various parts of their network. In addition, TNM has managed to attract 3 other large domestic customers that previously used our main competitor's product partially due to the functionality provided by ECXpert.

ECXpert has increased customer revenue by reducing the amount of time to isolate and repair network problems. Current estimates show that due to decreased network down time and reduced labor costs, savings at a typical U.S. network operation center are in the range of $500,000 and $1,000,000 a year.

The current version of ECXpert can correlate about 1000 alarms per hour with 10 correlation groups active. Although this meets customers current use of the feature package, users are becoming more sophisticated and are adding more and more correlation groups. ECXpert is currently running on a Tandem FT computer and is competing for resources with the rest of TNM. A future release will provide increased performance by allowing ECXpert to run on an adjunct processor.

Other enhancements currently under development include adding a graphics correlation window and replacing the relational database that stores the correlation tree with an object database. We are also working on a learning module to derive correlation groups automatically. For example, suppose in a number of 5 minute windows the messages A, B, C, D, and E occurred in sequence and they all had a common A Office or Z Office. The learning module could generate a correlation group with A causing B, B causing C, etc. We denote this as A→B→C→D→E. Now suppose there were other instances that consisted of (F, B, C, D, E). The learning module could now derive a correlation group with (A or F)→B→C→D→E.

This is a valuable feature as many of our customers do not know all the alarm types and how they should be correlated to network events. We do supply a default set of correlation groups, but the customer needs to configure and add rules to match their particular network needs. The search will be directed by meta-correlation rules written by the customer that will allow them to specify time windows, fields of interest, and strength (that is how many times must a pattern repeat before it is more than just a coincidence). The groups will then be generated automatically and presented to the users for modification, installation, and sometimes for deletion as chance patterns of messages can be grouped.

7 CONCLUSION

Some indication of the benefits of Prolog for this project can be gained by comparing it with another knowledge-based, network management feature package developed for TNM. This application analyzed error messages generated by the 5ESS switch and recommended repair procedures. C5 was used to implement rules collected from 2 experts from the New England Telephone Company. One problem limiting general deployment of this package is that the recommended repair procedures vary between customers. Companies often had different recommendations as to what to try first, what procedures are too risky, what actions are considered a breach of security, etc. Thus, although the system had a large amount of expertise, it was very inflexible and narrow. In contrast to Prolog, a meta-programming approach is not supported by C5. Although they used a pseudo-English description language to capture the experts knowledge, no compiler could be written and the hand encoding into C5 led to many misinterpretations and errors.

Meta-programming is a very powerful feature and useful technique that contributed to the success of ECXpert. Each customer can write their own correlation groups or modify the default groups we provide. They can then compile, test and use these groups in the field without having to interact with AT&T and request that we make the changes. Thus, new correlation groups can be added very quickly and the system can be configured to match each customers individual needs. Prolog not only facilitates the use of meta-programming, but it also allows changes to be dynamically linked with running processes. That is, there is no need to recompile the entire correlation process to use the new set of correlation groups. Nor is there even any need to stop the correlation process. Rather, correlation groups can be complied off-line and then linked dynamically with the running process.

Event correlation is not restricted to telecommunications, but is applicable to many other domains where order must be made of a large volume of related messages. I have spoken to people who have worked as air traffic controllers, power station operators, and chemical plant engineers. They all indicated their need to correlate large volumes of data collected from many pieces of equipment. Moreover, the knowledge required to group these messages together can also be represented as correlation tree skeletons. Thus, the correlation tree skeletons and the correlation algorithm used in ECXpert can be reapplied in many other domains, and should be included as a new generic task in problem solving (Chandrasekaran, 1986). Due to the importance of solving event correlation in leading our competitors and in its potential in other fields, a patent application with the specifics of my algorithm has already been filed by AT&T.

In conclusion, the multi-paradigm implementation provided a powerful environment that enabled us to combine the strengths of logic and object oriented programming. The user definability, dynamic linking, and the high level of abstraction provided by the correlation groups have been keys to success. Customers have a syntax that is powerful enough to configure the system the way they want using a language they can understand.

8 REFERENCES

Ambros-Ingerson, J. A. and Steel, S. (1988) Integrating Planning, Execution, and Monitoring, *Proceedings AAAI*, 83–88.

Abelson, H. and Sussman, G. (1985) Structure and Interpretation of Computer Programs, *MIT Press*, Cambridge.

Bobrow, D. and Stefik, M. (1986) Perspectives in Artificial Intelligence Programming, *Readings in AI and Software Engineering*, Morgan Kaufmann, California.

Chandrasekaran, B. (1986) Generic Tasks in Knowledge Based Reasoning: High Level Building Blocks for Expert System Design, *IEEE Expert*, **1(3)**, 23–30.

Clancey, W. (1983) Heuristic Classification, *AI Journal*, **27**.

Fikes, R.; Kehler, T. (1985) *Communications of the ACM*, **28**, 904.

Kowalski, R. (1979) Logic for Problem Solving, North-Holland, Amsterdam.

Nerys, C. (1993) The Complete Diagnostic Tool: Total Network Management, *Network Edge - AT&T*, 18–22.

Nygate, Y. (1994) ASPEN - Structuring Design of Complex Knowledge Based Systems, *Ph.D. thesis*, Case Western Reserve University.

Nygate, Y. and Sterling, L. (1993) ASPEN - Designing Complex Knowledge Based Systems, *10th Israeli Symposium on Artificial Intelligence*, 51–60.

Shortliffe, E. H. (1976) MYCIN: Computer-Based Medical Consultations, *Elselvier*, New York.

Sterling, L. (1990) Meta-Programming in Logic Programming Tutorial Notes, *Meta 90*, Leuven, Belgium.

Sterling, L. and Shapiro E. (1986) The Art of Prolog, *MIT Press*, Cambridge MA.

Yalçinalp, L. Ü. (1991) Meta-Programming for Knowledge-Based Systems in Prolog, *Ph.D. thesis*, Case Western Reserve University.

9 BIOGRAPHY

Yossi Nygate has been employed by AT&T Bell Labs for the past ten years. He has been responsible for developing telecommunication network management systems integrating C++, C, and Prolog for domestic and international customers. He received his Ph.D. in computer science from Case Western Reserve University in 1994. The focus of his research was on problem solving systems integrating multiple techniques. He received his M.Sc. in computer science from the Weizmann Institute of Science in 1985 in the area of Expert Systems. His current areas of interest include practical applications of AI, planning, and automated learning.