A Framework for Event Correlation in Communication Systems

Mouayad Albaghdadi[1], Bruce Briley[2], Martha Evens[3], Rafid Sukkar[4], Mohammed Petiwala[1], and Mark Hamlen[1]

[1] Motorola Inc., 1301 E. Algonquin Road, Room # 3346, Schaumburg, IL 60196, USA
{albaghdadi, Mohammed.H.Petiwala, CSPP03}@motorola.com
[2] Motorola Inc., 50 Northwest Point Blvd., Elk Grove, IL 60007, USA
bbriley@ieee.org
[3] Illinois Institute of Technology, CS Department, 10 West 31st Street, Chicago, IL 60616, USA
evens@iit.edu
[4] Lucent Technologies, 2000 N. Naperville Road, Naperville, IL 60566, USA
sukkar@lucent.com

Abstract. This paper introduces a framework for event correlation in communication systems. We will show how the concept of a class in object-oriented methodology can be used to provide scalability to the framework. Events and system topology information are combined to generate the causal information needed for correlation. Geometric representation of codewords is used to overcome the noise factor. Temporal reasoning is explored to reduce noise and increase the number of event patterns that can be detected. The framework has been applied to a wireless communication system.

1 Introduction

Event correlation in communication systems is the process of analyzing the information (events) received to determine a higher-level picture of the system behavior. The stream of events arriving at the centralized location is the system operator's main view of the managed system. The operator uses this information in a variety of ways. He can check the status of periodic test results performed on certain nodes, look for small sets of events indicating the failure of a node, reconfigure a remote site (in a wireless system) and wait for a reaction (in terms of events) as to how successful the reconfiguration process was, etc. A more typical task of event correlation in large systems is to correlate events from multiple nodes throughout the system to determine a higher-level picture of the system. Obviously, such nodes are related in some fashion and information from all of them can be combined to give a complete picture.

To meet users' demands for access to information, today's systems are in constant change. In addition to the large size and increased complexity, a system's topology changes frequently. As the system grows in size and complexity, more events will be generated that need to be analyzed and acted upon by the system's operator.

E.S. Al-Shaer and G. Pacifici (Eds.): MMNS 2001, LNCS 2216, pp. 271–284, 2001.

Moreover, the change in topology changes the relationships between nodes in the system. Thus, correlating events across nodes becomes a time dependent task that the operator has to keep track of.

A final challenge to event correlation in communication systems is the factor of noise. Noise problems can be attributed to one or more of the following: (1) events lost due to links down, which prevent events from arriving at their centralized location; (2) limited resources at different levels leading to dropped events; (3) spurious events; (4) related events that are delayed and/or arriving out of sequence; and (5) unrelated events arriving at the same time a pattern is being formed.

The system operator's main task is to operate and maintain the system. Such a task becomes very difficult to perform effectively given the above conditions. The result is that important information about the health of the system is misinterpreted or totally ignored.

A primary use of event correlation in communication systems is the detection and identification of faults. These two functions are part of fault management, which consists of a set of functions that enables the detection, isolation, and correction of abnormal operation of the system [1].

Due to the importance of event correlation, researchers have applied a variety of techniques that span many fields in their investigation of this problem. Artificial Intelligence (AI) techniques have been used by [2, 3, 4, 5, 6, 7]. In [8, 9, 10], the authors utilize the causal relationships between events to generate a causality graph. A coding technique is then used to identify event patterns. In [11], the authors used the causality graph outlined in [9] to model the problem, and applied algebraic operations of sets to identify event patterns. In [12], graph theory and propositional relations between events were used. Finite-State Machines (FSM) and probabilistic FSMs have been used by [13, 14, 15, 16]. Probabilistic models have been used by [17, 18]. Petri Nets have been used by [19, 20].

In this paper, we will investigate the problem of event correlation. We will exploit the object-oriented concept of a class to provide scalability to our approach. Techniques using graph and coding theories are presented to correlate events in distributed, dynamic, and noisy communication systems. Concepts of temporal reasoning and their application to event correlation are discussed. These techniques have extended the work of [9] in many ways. This investigation is being applied to a wireless communication system.

This paper is organized as follows: Section 2 describes the system alphabet. Section 3 presents the correlation language of the system. Section 4 gives the details of the event correlation model. Section 5 deals with related research. Section 6 and 7 contain our conclusion and references respectively.

2 The System Alphabet

The structure of a typical event is shown in Fig. 1 below. Here, the event type and error code together indicate the reason for the event. The network element and object names identify the source of the event. The reason field provides additional information about the reason for the event in some cases. The time field indicates when the event was generated.

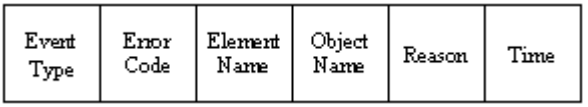

Fig. 1. Event structure

We can think of all the possible events generated by a system as its alphabet. Let's define the set of all events (the alphabet) generated by node i as Σ_i. The total set of all the Σ_i in a system with n nodes is $\Sigma_T = \Sigma_1 \cup \Sigma_2 \cup ... \cup \Sigma_n$. It is clear that the system's alphabet will increase linearly as the number of nodes in the system increases.

Communication systems can be characterized as a collection of objects organized in a way to provide specific services to the end user. An object can be defined as a distinct entity in the system. It can be a hardware entity, a software process, a communication link, etc. A node consists of one or more objects. A system consists of a finite set of node classes (e.g., router, server, etc.). For each of these classes, there exist one or more instances of such a class.

Given the structure of the events and the environment in which they exist, it is obvious that nodes belong to certain classes and each class has the same alphabet. This can be seen in Fig. 2 where we divided an event into two parts: topology information and event information. The event information is the class's alphabet.

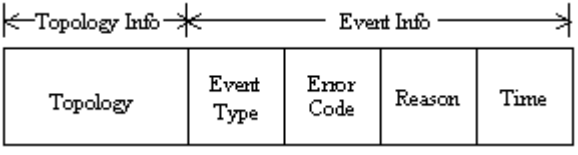

Fig. 2. Class event structure

The restructuring of events in this fashion shows clearly that the system's alphabet can be reduced to that of classes of nodes and objects within nodes. Therefore, a system's alphabet is dependent only upon the types of nodes and objects within nodes and not upon the instances of such entities.

3 The Language

We define a pattern (word) as a concatenation of events, e.g., $p_1 = e_1 e_2 e_3$ without regard to the arrival sequence. Therefore, $e_1 e_2 e_3$, $e_2 e_1 e_3$, $e_3 e_2 e_1$, etc., are the same pattern. Furthermore, the multiplicity of an event within a pattern is equivalent to a single occurrence. Let Σ^k denote the set of words with length k. Thus, $\Sigma^k = \{w | w$ is a word over Σ and $(|w| = k)\}$. For example, if $\Sigma = \{e_1, e_2, e_3\}$,

$\Sigma^2 = \{e_1e_2, e_1e_3, e_2e_3\}$. Let $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup ... = \bigcup_{k=1}^{\infty} \Sigma^k$. Note that Σ^+ is the set of words that might be constructed from one or more events of Σ, and is the largest set of possible words we might observe from an entity being modeled. Let $L(C)$ be the language of a class, e.g., an object. A language is a set of words that can be constructed from the alphabet. Therefore, $L(C)$ is a subset of Σ^+.

4 The Model

An event pattern may consist of tens or hundreds of events. These events are the contribution of many objects throughout the system. The number of events generated by each object and the number of objects involved in a pattern are dependent on the type of the pattern and the size and complexity of the system.

Given this framework, the correlation process is divided into two phases: local correlation, which is an object level correlation, and global correlation, which combines the outcomes of the local correlation process in a way to determine a higher-level picture of the system behavior. This model is appropriate for the problem at hand. What we are saying is that objects generate events in two situations: (1) changes in the state of the object itself; and (2) the object's reaction to external changes. In both cases, an object's behavior is based on its own limited view of the system. The global correlation process collects and assembles these objects' views to determine the system view.

The correlation language can be modeled as a bipartite graph, where one set of nodes is the alphabet and the other is the words (patterns). Fig. 3 below depicts an example of such a graph. The figure shows three patterns and eight events, and the causal relationships between them. The causality is depicted by the arrow "$\rightarrow$" pointing from the pattern to the event.

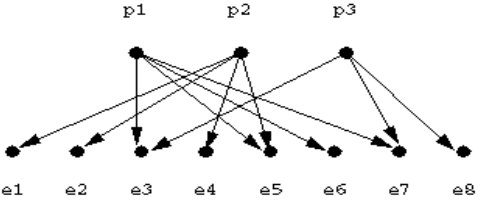

Fig. 3. Relationships between patterns and events

The figure above shows that patterns share common events. This is typical of real systems where subsets of the alphabet overlap but identify different patterns. Sometimes we are not that lucky and the same subset may indicate more than one pattern. In that case, the correlation result will have multiple outcomes.

There is more than one way to process the graph above. As an example, our initial but brief investigation was directed toward the use of neural networks using the

software package described in [21]. Our approach follows that of [9]. The language can be modeled as a set of patterns (a template) for an object class. This template can be seen as a matrix where the columns are the different patterns and the rows are the types of events making up the patterns. The above graph is shown below as a matrix, where there is a 1 in row i and column j *iff* there is an arrow from p_j to e_i.

	p1	p2	p3
e1	0	1	0
e2	0	1	0
e3	1	0	1
e4	0	1	0
e5	1	1	0
e6	1	0	0
e7	1	0	1
e8	0	0	1

Fig. 4. Correlation matrix

4.1 Pattern Identification in a Noisy Environment

Given the noisy environment in communication systems, there is a need for an algorithm to compensate for this inevitable problem. Coding and information theory is in large part concerned with noise when transmitting and storing information. Techniques for error-detecting and error-correcting codes are well established in this area and are suited to address this problem.

In [9], Hamming codes were to represent and identify (decode) event patterns. However, their work does not specify the type of the decoder being used. Our investigation utilizes the same technique and at the same time provides a suitable decoder for the problem at hand. First we start with some definitions of the principles of error-control coding.

A codeword is a fixed length vector of 0s and 1s. Fig. 4 depicts three codewords $p1$, $p2$, and $p3$. The Hamming distance weight $w(p_i)$ of codeword p_i is the number of 1s in p_i. The Hamming distance $d_{ij} = d(p_i, p_j)$ between codewords p_i and p_j is defined as the number of positions in which p_i and p_j differ. The greater this number, the greater the geometric distance between them. The distance between two codewords can be obtained by applying the modulo-2 addition $\oplus$ (the logical operator Exclusive-OR in Boolean Algebra) to the two codewords p_i and p_j as follows: $d_{ij} = w(p_i \oplus p_j)$. For example, the Hamming distance between the two codewords $p1$ and $p2$ in Fig. 4 is 6.

The power of a code has been stated in [22] as follows:

$$d_m = \begin{cases} e+1 & \text{error detection} \\ 2e+1 & \text{error correction} \end{cases} \quad (1)$$

Here a code that can detect up to e errors per word and correct up to e errors per word must have a minimum Hamming distance d_m. It is obvious that a code having all distinct codewords must have a minimum Hamming distance of at least 1.

Given this equation for a minimum Hamming distance, we can construct a code with different levels of resilience to noise. This concept can be used to tailor a given code to different operating environments, i.e., different systems facing different noise levels. It also helps optimize the size of the correlation matrix, which [9] refers to as the "codebook".

The following equation is our Hamming decoder:

$$\cos\theta = \frac{u \cdot v}{\|u\| \quad \|v\|} \quad (2)$$

This decoder corresponds to a geometric representation of two vectors u and v in n-space. Assume that the two vectors have been positioned so their initial points coincide. The angle between them satisfies $0 \le \theta \le \pi/2$. The term $u \cdot v = u_1 v_1 + u_2 v_2 + \ldots + u_n v_n$ is the dot product of the two vectors, while $\|x\| = \sqrt{x_1^2 + x_2^2 + \ldots + x_n^2}$ is the norm of a vector x. When $\cos\theta$ is equal 1, i.e., θ is zero, we have a perfect match while for a value of 0 $\left(\theta = \pi/2\right)$ the two vectors are complete opposites. To compensate for noise, a pattern match is declared once $\cos\theta$ exceeds some threshold, e.g., $\cos\theta \ge 0.9$.

4.2 Local Correlation

Our approach is to identify the correlation matrices for all object classes in the system under study. As events arrive at the Operations and Maintenance Center (OMC), or are read from the event log offline, we use the topology information part of the event to identify the source of the event. The topology information provides us with the node instance and the object name of the source of the event as well as the node's hierarchy in the system. The event will be stored in the object's queue in a FIFO fashion. If this is the first event generated by the object, a new queue is created and the event is placed first in the queue. The event is assigned a number based on the object's alphabet and is labeled as an uncorrelated event. This means that the event has not been identified as part of a pattern yet. The queue length is a system dependent parameter. We have used a length of twenty based on our system's empirical evidence. Using this technique, old events will be pushed out of the queue as more events arrive at the queue.

With the arrival of every new event in an object's queue, uncorrelated events are used to form a vector called a "running vector". A similarity measure is applied to the running vector and every vector (pattern) in the object's class template. The vector that is most similar to the running vector is declared as its match. All events that contributed to the pattern are labeled as "correlated" and can be removed from the queue. A local correlation result, i.e., a composite event, will be generated as a result of the match. If however, no match is found, no action is taken.

Defining bipartite graphs in local correlations start with the definition of event-pattern pairs. This knowledge is stored in text files, which are then read into the system during initialization.

In local correlation where an object class within a node class is the focus, each event-pattern pair consists of the three fields:

- LCEC: Local Composite Event Code which is the result of a local correlation;
- PEC: Primitive Event(s) Code is a set of primitive events represented numerically. This is a subset of the object's alphabet;
- LCED: Local Composite Event Description. It is a text description of the local correlation result. It is used by the human operator to describe the correlation result if needed.

To illustrate the above, we use Fig. 4 as an example. First, assign a unique LCEC value for each pattern as follows: p1 = 1, p2 = 2, and p3 = 3. Second, assign a unique PEC value for each event as follows: e1 = 1, e2 = 2, e3 = 3, e4 = 4, e5 = 5, e6 = 6, e7 = 7, and e8 = 8. Third, assign a unique LCED field to each pattern. This can be a simple description of the pattern, which can be used by the operator. Finally, each field is separated by a delimiter (e.g., vertical bar |).

1 | 3 5 6 7 | BSC link object is down

2 | 1 2 4 5 | BSC link object is up

3 | 3 7 8 | BSC link object is active.

4.3 Global Correlation

We define a correlation zone (CZ), as a logical entity that represents a collection of objects. A CZ is modeled as a bipartite graph where one set of nodes represents the correlation patterns and the other represents the composite events. A CZ subscribes to the composite events, which are received from objects within the system. The bipartite graph must consist of one or more pattern nodes and the composite events must be from two or more objects within the system. The objects can be located within the same node or different nodes or a combination of the two. A CZ is represented by a queue, which receives composite events from the objects it subscribes to. Like the local correlation decoder, the CZ uses the same approach to identify arriving patterns. The CZ is identical to the concept of an object and its approach to event correlation. The only difference is that a CZ does not generate events and therefore it does not have its own alphabet. Instead, the user defines its alphabet and its language according to known causal relationships. The global correlation process is recursive. This means that a CZ class can be made of smaller set

of CZs. This concept fits well given the hierarchical nature of communication systems.

While the events in local correlation are the object's alphabet, the global correlation's alphabet is made up of the composite events subscribed to by the CZ based on casual relationships. Therefore, a CZ can be represented by any arbitrary bipartite graph that consists of patterns and composite events. Our objective in this step of the correlation process, however, is to address the scalability issue while finding a solution that works. This can be achieved by using the class concept we explained in the local correlation step.

In wireless communication systems the concept of hierarchy is evident. A system consists of thousands of base stations and hundreds of site controllers, both of which are located remotely. Other types of nodes including the OMC are located in the central office. These nodes belong to a few node classes, which make them ideal candidates for object-oriented methods.

Defining bipartite graphs in global correlations is similar to that of local correlation. It starts with the definition of event-pattern pairs. This knowledge is stored in text files, which are then read into the system during initialization. However, there are two problems that need to be addressed.

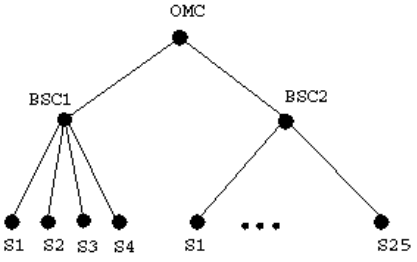

Fig. 5. A small segment of a system

First, patterns of a CZ are dependent on the system's size and complexity. Therefore, the bipartite graph of a CZ must be built dynamically with input provided by the topology information collected from the OMC. To illustrate this concept, consider, for example, Fig. 5, where a small segment of a wireless system is depicted. Here, BSC1 controls four sites while BSC2 controls twenty-five sites. There is an X.25 connection between each site and the OMC that passes through a BSC. Moreover, an X.25 connection exists between every BSC and the OMC. If we consider a BSC outage pattern to be all the X.25 link failure events between the OMC and the sites controlled by that BSC, then BSC1 has a different outage pattern than BSC2. This information can only be captured by accessing the topology information stored at the OMC.

Second, we need a way of collecting all the composite events subscribed to by a CZ in a single location for correlation. This can be achieved through the following steps:

- During system initialization, for each global pattern, identify the node(s), the objects, and the local result LCEC(s) that make up the global pattern;

- Send a message to each object identified above indicating the need to subscribe to the specified LCEC(s);
- Objects receiving such messages will store the destination address of the CZ along with the corresponding LCEC in a table;
- During normal operation, once an object detects a pattern, an LCEC is assigned to it. The object will search through its subscription table to see whether it needs to be routed to a destination for global correlation or not.

Each composite event is defined as follows:

1. CZ Class: The name of the class that the CZ belongs to;
2. Object Class: The object class within the CZ class;
3. LCEC: Local Correlation Event Code of the object's result.

Using topology information, classes are converted into instances with which a CZ has a causal relationship. Once this conversion is completed, a global bipartite graph is generated, which is similar to that of a local correlation. Like the local correlation step where a code is assigned for each pattern (LCEC), we will assign a code for a global pattern and call it a Global Correlation Event Code (GCEC).

The relationship between an object and the global correlation patterns may be one-to-one or a one-to-many relationship. This means that a single LCEC can be sent to one or more CZs. The type of relationship is governed by the causal relationships between them.

An example illustrating these concepts appears in Fig. 6, which depicts a segment of a wireless system. There is an X.25 link between every S node and the OMC which passes through a BSC. Let us define a CZ class to be an S node. We define an S node failure as one of the patterns within this CZ.

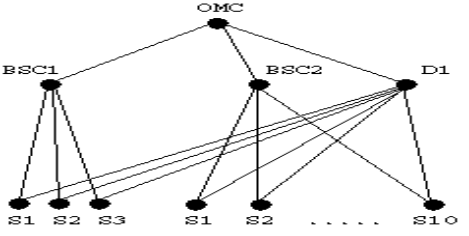

Fig. 6. Simple segment example

The composite events making up this pattern are as follows: a link object failure in the controlling BSC (between BSC and an S node), a BSC controller object failure which controls a set of S nodes, a link object failure between a D1 and an S node, and the X.25 link object failure in the OMC that controls a specified S node. The pattern can be specified as follows:

S| BSC-S-Link| 1

S| BSC-Cntrl||1

S| D-S-Link|1

S| OMC-S-X25|1

The next step is to convert this template pattern into an instance for a specified CZ. Let us take the S1 node as an example. The template pattern becomes:

S1|BSC1-S1-Link|1

S1|BSC1-Cntrl|1

S1|D1-S1-Link|1

S1|OMC-S1-X25|1

The CZ for S1 then sends a message to subscribe to all the objects in the pattern. It is important to note that the BSC1-Cntrl has a one-to-many relationship with the S node class. Therefore, when the controller object generates an LCEC of 1, it will send it to all the S nodes it controls, i.e., CZ of S1, S2, and S3, even though CZ S2 and S3 do not need it. This is not a problem to be concerned about since S2 and S3 will flush out these unwanted events either due to the limited size of the queue holding the events or through temporal reasoning.

4.4 Temporal Reasoning

The temporal relationships between events are as important as causal relationships. We plan to utilize the temporal relationships in two different areas, (1) reducing noise and therefore improve the accuracy of the correlation results; and (2) solving the problem of variable length codes.

Local and global entities are represented by queues where events arrive for correlation. Spurious and delayed events among others are the cause of noise. The events stay in the queue until they are flushed out by the FIFO effect. This FIFO effect is helpful but not enough to solve the noise problem completely. Next, we introduce ways to incorporate the temporal relationships between events to improve the noise problem.

There are three levels at which a temporal relationship can operate: (1) event level; (2) pattern level; and (3) queue level. We start with the event level temporal relationships. Here, events are considered as independent entities. Only the aging concept will be considered at this level. Three different types of aging techniques can be employed: (1) Threshold: an event will be active until a predefined value is reached when it is considered obsolete and should be removed from the queue. This behavior can be modeled as a step function; (2) Mathematical functions: an event is assigned a weight based on its age in the queue. Upon arrival in the queue, an event has a weight of 1. This weight decreases as time goes by until the weight becomes zero and the event is removed from the queue. The weight value follows the behavior of some function. Finding the best model for event aging is not simple and is system dependent. Simple functions such as linear or truncated exponential can be easily implemented and might be a good starting point.

If we treat all the different events within a queue equally, then the same techniques described for an event level can be applied to a queue. As an example, consider the

threshold level technique. Here, we assign a threshold value for a queue where any event older than that value will be considered obsolete and must be removed. The queue level temporal relationships can be seen as a simplified version of event level temporal relationships.

The last technique uses the pattern level temporal relationships. Here, every set of events that make up a pattern is treated differently. This is true of the same event belonging to different patterns. This level of temporal relationships is the most effective in dealing with the noise problems in communication systems because it treats the temporal constraints of every pattern differently.

Pattern level temporal relationships require more than a simple aging technique. There is a need for a temporal language to allow for the specification of a variety of temporal relationships between a set of events. In [23], an overview of existing temporal logic programming languages was given. While [24], provided composite event specification language that can be used to express complex temporal relationships between events. For now, we will define the following single operator: All events comprising a pattern must arrive within a specified period of time. This temporal constrained operator is useful in eliminating old and unrelated events.

So far we have been concerned with using temporal logic to reduce noise. Another important use of temporal logic is in the area of variable-length codes. Consider for example the following scenario: A periodic process runs every 30 minutes and if successful, it generates a pattern of 3 events: Ready, Start, and Complete. If the process fails, only one or two events are generated, i.e., Ready or Ready and Start are generated. The problem of distinguishing between the two patterns is that the failure pattern is a prefix of the success pattern; that is, the failure pattern is the same as the beginning of the success pattern. This problem is similar to the variable-length code problem – the receiver does not know when one pattern is over without looking further.

To illustrate the problem further, consider the following example which is outlined in [25]. Let's assume we have four symbols to be coded as follows: s1 = 0, s2 = 01, s3 = 011, and s4 = 111. Assume that the receiver receives the string 0111 . . . 1111. It can only be decoded by first going to the end and then identifying groups of three 1s as each being an s4 until the first symbol is reached. Only then can the first symbol be identified as s1, s2, or s3.

Waiting until the end of the string does not work in a noisy environment, i.e., with delayed, missing, or spurious events. Temporal logic can be used to address this problem more effectively. We can devise a pattern level temporal operator where once a pattern is detected, it is activated some time later if the same conditions still exist, i.e., the same events that triggered the pattern detection still in the queue. This operator is similar in concept to a watchdog timer activation and deactivation. Say we have two patterns p1 and p2 and assume that p1 is a prefix of p2. When p1 is detected, the watchdog timer is activated and is set to a predefined time. If p2 is detected before the watchdog timer expires, it is disabled and p2 is declared as the detected pattern. If, on the other hand, the watchdog timer expires, p1 is then declared as the detected pattern. This concept can be extended to multiple patterns where each is a prefix of the other. Such an operator will help terminate an arriving pattern properly for positive identification.

5 Related Research

As we mentioned earlier, many researchers have investigated this problem applying a variety of techniques (see section 1). This section is intended to only describe previous research that we used to build our work on. We hope that this will give credit to previous work we used and highlight our contribution.

The work of [13] was based on fault detection using a Finite-State Machine (FSM) as a model of the system. In the system language section, we have modified some of the definitions used by them to fit our model.

In [8, 9], a graph theoretic approach was applied by utilizing the causal relations among events. The authors describe a distributed event correlation system and at the same time address some major issues in event correlation, i.e., scalability, resilience to noise, and generality.

The notation $p \rightarrow s$ is used to denote causality of event s by event p. Causality is defined as a partial order relation. The relation $\rightarrow$ is described by a causality graph, which is a directed graph whose nodes represent events and whose edges represent causality. Each event in this causality graph is either a symptom event or a problem event. This graph is then pruned (all indirect symptoms and cycles are removed) to produce a correlation graph. Finally, a codebook with a predefined minimum Hamming distance is extracted from the correlation graph.

These graphs are generated automatically based on a specification model of the network at hand. The specification model is written in an object oriented modeling language called MODEL (Managed Object Definition Language). MODEL is an extension of the CORBA IDL language. It adds new syntactic constructs to specify semantics that cannot be specified in CORBA IDL: relationships, events, problems, and causal propagation.

6 Conclusion

We have introduced a framework for event correlation in communication systems. We showed how the concept of class in object-oriented methodology is used to provide scalability to our approach. Graph and coding theories are used for correlation. Finally, temporal reasoning was investigated for this framework.

We believe that this work has extended the research done by [8, 9] in many ways. We replaced the MODEL language with a simple free text causal language. We introduced a new decoder for pattern identification. Entities and their queues are created only if they generate events and are deleted once their queues are empty. This helps reduce memory requirements and decrease system complexity. Temporal relationships have been shown to be effective alongside the causal relationships between events.

References

1. ANSI T1.215, OAM&P - fault management messages for interface between operations systems and network elements, 1994.

2. Appleby, K., Goldszmidt, G., Stiender, M., "Yemanja - A Layered Event Correlation Engine for Multi-domain Server Farms," Integrated Network Management VII, pp. 329-344, May 2001.
3. Jakobson, G., Weissman, M., Brenner, L., Lafond, C., Matheus, C., "GRACE: Building Next Generation Event Correlation Services," Proceedings of IEEE/IFIP Network Operations and Management Symposium, pp. 701-714, April 2000.
4. Jakobson, G., Weissman, M., "Alarm Correlation," IEEE Network, Vol. 7, No. 6, pp. 52-59, Nov. 1993.
5. Sheers, K., "HP OpenView Event Correlation Services," Hewlett-Packard Journal, pp. 31-42, Oct. 1996.
6. Wietgrefe, H., Tuchs, K., Jobmann, K., Carls, G., Frohlich, P., Nejdl, W., Steinfeld, S., "Using Neural Networks for Alarm Correlation in Cellular Networks," International Workshop on Applications of Neural Networks in Telecommunications, pp. 248-255, June, 1997.
7. Weiss, G., Eddy, J., Weiss, S., "Intelligent Telecommunication Technologies, " in Knowledge-Based Intelligent Techniques in Industry, Jain, L., Johnson, R., Takefuji, Y., Zadeh, L., Editors, pp. 251-275, CRC Press, Boca Raton, FL, 1999.
8. Kliger, S., Yemini, S., Yemini, Y., Oshie, D., Stolfo, S., "A Coding Approach to Event Correlation," In A.S. Sethi, Y. Raynaud, and F. Faure-Vincent, editors, Proceedings of the Fourth IEEE/IFIP International Symposium on Integrated Network Management, pp. 266-277, Chapman and Hall, London, UK, May 1995.
9. Yemini, S., Kliger, S., Mozes, E., Yemini, Y., Oshie, D., "High Speed and Robust Event Correlation," IEEE Communications Magazine, Vol. 34, No. 5, pp. 82-90, May 1996.
10. Albaghdadi, M., Hood, C., Hamlen, M., "A Framework for Distributed Event Correlation," Proceedings of the 17th IASTED International Conference - Applied Informatics, pp. 467-470, Innsbruck, Austria, Feb. 1999.
11. Lo, C., Chen, S., "Robust Event Correlation Scheme for Fault Identification in Communications Network," GlobeCom, pp. 3745-3750, 1998.
12. Hasan, M., Sugla, B., Viswanathan, R., "A Conceptual Framework for Network Management Event Correlation and Filtering Systems," Sixth IFIP/IEEE International Symposium, pp. 233-246, 1999.
13. Wang, C., Schwartz, M., "Fault Detection with Multiple Observers," INFOCOM, pp. 2187-2196, 1992.
14. Bouloutas, A., Hart, G., Schwartz, M., "Simple Finite-State Detectors for Communication Networks," IEEE Transactions on Communications, Vol. 40, No. 3, pp. 477-479, March 1992.
15. Bouloutas, A., Hart, G., Schwartz, M., "Fault Identification Using a Finite State Machine Model with Unreliable Partially Observed Data Sequences," IEEE Transactions on Communications, Vol. 41, No. 7, pp. 1074-1083, July 1993.
16. Rouvellou, I., Hart, G., "Automatic Alarm Correlation for Fault Identification," INFOCOM, pp. 553-561, 1995.
17. Deng, R., Lazar, A., Wang, W., "A Probabilistic Approach to Fault Diagnosis in Linear Lightwave Networks," IEEE Journal on Selected Areas in Communications, Vol. 11, No. 9, pp. 1438-1448, Dec. 1993.
18. Dawes, N., Altoft, J., Pagurek, B., "Network Diagnosis by Reasoning in Uncertain Nested Evidence Spaces," IEEE Transactions on Communications, Vol. 43, No. 2/3/4, pp. 466-476, Feb./Mar./Apr. 1995.
19. Boubour, R., Jard, C., Aghasaryan, A., Fabre, E., Benveniste, A., "A Petri Net Approach to Fault Detection and Diagnosis in Distributed Systems. Part I: Application to Telecommunication Networks, Motivation, and Modeling," Proceedings of the 36th Conference on Decision and Control, pp. 720-725, San Diego, CA, Dec. 1997.

20. Aghasaryan, A., Fabre, E., Benveniste, A., Boubour, R., Jard, C., "A Petri Net Approach to Fault Detection and Diagnosis in Distributed Systems. Part II: Extending Viterbi algorithm and HMM Technique to Petri Nets," Proceedings of the 36th Conference on Decision and Control, pp. 726-731, San Diego, CA, Dec. 1997.
21. Stuttgart Neural Network Simulator (SNNS). University of Stuttgart, Institute for Parallel and Distributed High Performance Systems (IPVR), [cited 7 September 2000]. Available from http://www.informatik.uni-stuttgart.de/ipvr/bv/projekte/snns/snns.html.
22. Stremler, F., Introduction to Communication Systems, Third Edition, Addison Wesley, Reading, MA, 1990.
23. Orgun, M. A., Ma, W., "An Overview of Temporal and Modal Logic Programming," in Temporal Logic, First International Conference, ICTL 94, pp. 445-479, Springer-Verlag, Bonn Germany, July, 1994.
24. Liu, G., Mok, A. K., Yang, E. J., "Composite Events for Network Event Correlation," Integrated Network Management VI, pp. 247-260, May 1999.
25. Hamming, R., Coding and Information Theory, Second Edition, Prentice-Hall, Englewood Cliffs, NJ, 1986.