Event Modeling with the MODEL Language*

D. Ohsie[†]
Columbia University
Department of Computer Science
New York City, NY 10027
ohsie@cs.columbia.edu

A. Mayer[‡]*, S. Kliger, S. Yemini*
System Management Arts (SMARTS),
14 Mamaroneck Ave,
White Plains, NY 10601
alain, kliger, yemini@smarts.com

Abstract

Event modeling is an essential component of event correlation systems; this paper introduces the MODEL language, which comprises the event modeling component of SMARTS' InCharge™ event correlation system. We demonstrate the features of the MODEL language through examples from the multimedia Quality of Service (QoS) domain. In addition, we provide a comparison of MODEL with the event modeling capabilities of other event correlation systems; we demonstrate that MODEL generalizes the capabilities of other systems and is more flexible.

Keywords

Event correlation, event modeling, multimedia.

1 INTRODUCTION

Network management consists mainly of monitoring, interpreting, and handling of events or exceptional condition in the operation of the network. Event correlation is the process of automatically grouping related events based on their underlying common cause, thereby compressing the event stream and identifying potentially hidden problems. NetFACT (Houck et al. 1995), SINERGIA (Brugnoni et al. 1993), IMPACT (Jakobson and Weissman 1995), ECXpert (Nygate 1995) and the authors' own InCharge™ (formerly DECS) (Yemini et al. 1996) are all examples of such systems. An event correlation system consists of two basic components: an

*The process for event correlation and problem reporting described in this paper is covered by U.S. Patent No. 5,528,516.
This research was supported in part by Air Force Contract No. F30602-95-C-0262.
[†]This author's research was supported in part by NSF grant IRI-94-13847.
[‡]This authors current affiliation is Bell Labs / Lucent Technologies, 600 Mountain Ave, Murray Hill, NJ 07974, alain@research.bell-labs.com

event definition and propagation model (or simply *event model*), and a *reasoning algorithm.* The event model describes the underlying system, while the reasoning algorithm processes incoming events and correlates based on the knowledge contained in the event propagation model. The event model in turn consist of a *class-level* model, and a run-time *object topology.* The class-level model describes the general rules for propagating events from objects of one class to another, while the object topology describes a particular instantiation of the run time model which reflects the current state of the actual system.

As an example of event modeling, consider the scenario in Figure 1 from the Multimedia Quality of Service (QoS) domain. Here, a video sender, an electronic classroom located on the local area network LAN2, wishes to transmit some live video to a receiver located on LAN1 using the video tool `vic` (McCanne and Jacobson 1995a) which utilizes the UDP transport protocol. The UDP connection transports IP packets through routers D, C, B, and A which connect the LAN domains through a router backbone domain. The router backbone domain uses physical-layer wide-area network (WAN) domains. Similarly, an audio sender, Internet phone, on LAN4 wishes communicate with a receiver on LAN3, using the audio-tool `vat` (McCanne and Jacobson 1995b). Its IP packets are routed via F, C, B, and E. These transmissions, plus other, unrelated traffic cause the rate of packets

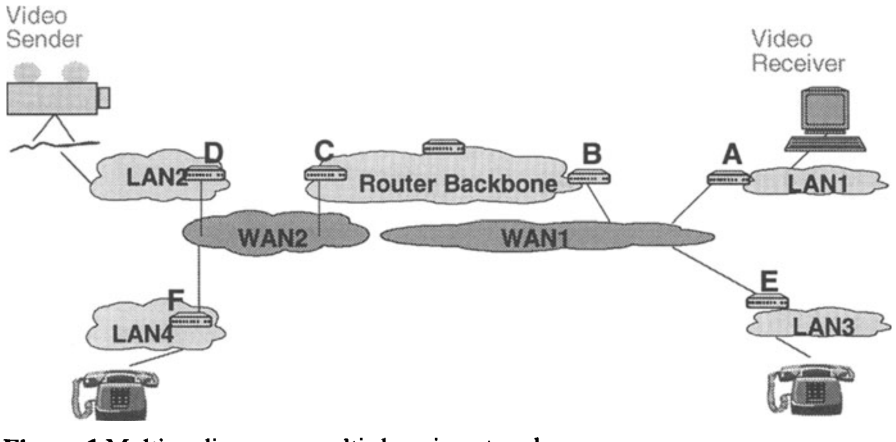

Figure 1 Multimedia over a multi-domain network.

arriving at C to be too high. Consequently the buffer at C overflows, causing the multimedia transmissions to lose packets. Note that congestion of this nature is the most common cause of packet loss on the Internet. The packet losses at router C will propagate to all UDP connections which router C is a part of. Since UDP does not retransmit lost packets, these losses will in turn propagate to the multimedia transmissions and hence the quality at the receiver may become unacceptable.

The *class-level event model* for the scenario described above consists of the following: a definition of the "poor video quality" event, a rule describing the propagation of router congestion to packet loss and then to poor video quality, and optionally a "high packet loss" event at the router level. The *object* topology consists of the individual routers and multimedia applications and their relationship in the underlying network. A *reasoning algorithm* would infer the presence of the congestion problem based on the poor video and audio quality and the event model illustrated.

In previous work (Kliger et al. 1995), we described the coding approach to event correlation which is the reasoning algorithm of our InCharge™ event correlation system (Yemini et al. 1996). There, we showed how the symptoms of each problem in a modeled system could be treated as a code for that problem, and that elementary techniques from coding theory could be profitably applied to event correlation. That work presupposes that there is a causality graph which maps each problem to its immediate (possibly unobservable) causal effects and in turn to others, until an observable symptom event is caused. Thus, computing the code for a problem involves computing a closure over the causal relationships emanating from the problem.

In this work, we describe the MODEL language for the definition of object and event models. MODEL supplies an object-oriented data model complete with inheritance and overloading. It also provides instrumentation capabilities to automatically tie attributes in the model to SNMP MIB variables. More importantly, MODEL supplies two feature which are essential to event correlation. First, it provides a declarative specification of events in the form of boolean expressions over attributes in the object model. This allows the definition of events to be integrated into the model of the objects in which the event occurs. Second, MODEL allows the user to specify *local* event propagation rules in which we show how to construct the causality graph from the combination of the class-level event propagation model and the current object topology. Often, event propagation patterns depend heavily on the way in which objects are currently interconnected; changing the topology of the modeled objects will drastically alter the observed symptoms of a problem. We will show that MODEL's approach to defining event propagation is superior to the event modeling capabilities of existing systems, because it can handle topology dependent event propagation through the use of event overloading. In addition, MODEL is correlation algorithm independent, so it can actually be substituted for the event modeling subsystems of existing correlation systems to improve their generality and ease of use.

The rest of this paper is organized as follows. In Section 2, we will use scenarios from the multimedia QoS domain among others to provide a "description by example" of the MODEL language. In Section 3, we will outline the process of developing reusable event libraries in MODEL. Section 4 will provide a critical comparison of the MODEL language with the event modeling capabilities of other event correlation system.

2 THE MODEL LANGUAGE AND QOS MANAGEMENT

In this section, we give an in-depth “introduction by example” to the MODEL language. We begin with an example from the multimedia management domain. Consider again the example configuration of Figure 1 and the following scenario: Due to high traffic volume, router C experiences congestion. As a consequence, its buffers overflow and incoming IP packets get lost. Since the video and the audio receiver are endpoints of a UDP connection which is layered over router C, they both experience the same type of QoS violation: an average transmission rate that is drastically below tolerance. A correlator, using the knowledge provided by the corresponding model, should report a high probability that the problem causing these violations is located in the domain of the router backbone.

We will begin by considering a simple example of a causal relationship, that of congestion causing lost packets. First we must define what we mean by “high packet discards”. Let us assume that the router implements the IP protocol and is instrumented via SNMP. We can then measure the total number of discarded packets by querying the SNMP MIB-II variables ipOutDiscards and ipInDiscards:

```
interface IPRouter: IP
{
    instrumented attribute long ipInDiscards;
    instrumented attribute long ipOutDiscards;
    attribute long discardsThreshold;

    event PacketDiscardsHigh "The level of discarded packets is high" =
    (delta   ipInDiscards   +   delta   ipOutDiscards)   /   delta   _time   >
    discardsThreshold;

    instrument SNMP;
}
```

In this example, the `attribute` statements defines measurable properties of the IP protocol entity. The `event` statement defines the circumstance under which the event can be said to have occurred. In this case, the event `PacketDiscardsHigh` will be deemed to have occurred whenever the sum of the changes `ipInDiscards` and `ipOutDiscards` per time exceeds a threshold. The `delta` keyword indicates that the difference between the new and old values of the attribute are desired. The `_time` keyword refers to the time at which samples are taken. Thus this event is triggered when the discard rate reaches the threshold.

Here we digress for a moment to reflect on the relationship between MODEL and SNMP. The ipInDiscards and ipOutDiscards attributes are automatically instrumented via SNMP; no additional programming is required to keep these attributes updated with current values. In addition, a utility program called *mib2model* can be used to parse SMI MIB definitions and generate the corresponding MODEL classes automatically. Thus all features of the MODEL language essentially extend the functionality of the underlying SNMP MIBS. This approach meshes well with the SNMP philosophy; the underlying device must implement only the simple SNMP protocol and can thus concentrate its resources on its task (in our example, routing packets). The event management system provides higher level services using dedicated management resources. MODEL enables the event modeler to ignore this distinction and concentrate on simply modeling the events without regard to who supplies the information. In our example, we have effectively extended the power of the standard SNMP MIB to include our newly defined event.

Now, let us return to modeling the congestion problem at the router. We want to express the fact that there is a causal relationship between the congestion problem and the high packet discard event (with probability 1.0):

```
problem Congestion "High congestion" = PacketDiscardsHigh 1.0;
```

This line would be added to the MODEL class definition above. Note that this is a semantic declaration in the form of a rule; however, it does not have any specific algorithmic or operational meaning. It simply expresses the fact that there is a causal relationship between these two events. The inclusion of the problem and symptom in the scope of a single class obviates the need to write the rule as follows:

```
Congestion(IPRouter(X)) -> PacketDiscardsHigh(IPRouter(X));
```

We have modeled a local symptom which indicates the problem of Congestion. However, we would also like to relate the problem to the other observed symptoms at the multimedia application level. In this way, anomalies observed at the multimedia level can be correlated with the problem detected at the lower level.

Problems in one object propagate to related objects via *relationships*. In our example, the Congestion problem would propagate to higher level connections which are layered over the congested IP node. Thus we would add the following statement to indicate the relationship between IP nodes and connections:

```
relationshipset Underlying, TransportConn, LayeredOver;
```

The keyword `relationshipset` indicates that many connections may be layered over a single IP node. Now, we would like to express the fact that the congestion problem causes both the local symptom PacketDiscardsHigh, and propagates those discards as losses in the higher level connection:

```
problem Congestion "High congestion" =
            PacketDiscardsHigh 1.0, ConnectionPacketLossHigh 0.8;

propagate symptom ConnectionPacketLossHigh =
            TransportConn, Underlying, PacketLossHigh;
```

Note that we have added the symptom ConnectionPacketLossHigh to Congestion problem and that we have used a causal probability of 0.8, where a value of 1.0 indicates complete certainty. This indicates that congestion at the IP node may not cause packet losses on all connections above it, depending on the circumstance surrounding the congestion; we would not want to rule out congestion simply because a single connection which is layered over the node is not experiencing problems.

The `propagate symptom` statement says that the symptom ConnectionPacketLossHigh refers to an event in a related object, namely the event PacketLossHigh in any TransportConn which layered over this IP node. Now, we will continue the example by presenting the MODEL code which further propagates the problem to its observable symptom in the multimedia layer:

```
interface TransportConn
{
    propagate symptom PacketLossHigh =
                Port, ConnectedTo, PacketLossHigh;
}

interface UDPPort: Port
{
    propagate symptom PacketLossHigh =
                Appl, Underlying, PacketLossHigh;
}


interface MM_InPort: Appl
{
  instrumented attribute long MinRate;
  instrumented attribute long MaxRate;
  instrumented attribute long MsgCounter;
  instrumented attribute long ActTime;

  computed attribute ActualRate = (MsgCounter)/(_time - ActTime);

  event BadRate = (MinRate > ActualRate) || (ActualRate > MaxRate);

  problem PacketLossHigh = BadRate 1.0;
}
```

Note that a TransportConn simply propagates the packet losses to the Ports to which it is connected; a UDPPort (which, being a subclass of Port, inherits from Ports) in turn propagates the packet losses to Applications which are LayeredOver the port. For simplicity, the relationships which are utilized for this propagation, ConnectedTo and Underlying, are not defined here. Typically they would be inherited from generic link and node classes in the Netmate hierarchy, which is described in Section 3.

The multimedia receive port, MM_InPort, is a subclass of Appl. Therefore, it receives, via inheritance, the PacketLossHigh symptom from the UDP_Port which it is LayeredOver. The PacketLossHigh event in the MM_InPort has a single locally defined symptom, thus we again utilize the *problem* statement to define its symptom. In this case, PacketLossHigh causes the observable symptom BadRate, which indicates the reception rate is out of tolerance. Since this symptom is observable, it is defined using the *event* statement and an expression to detect the symptom. This example also demonstrates the use of expressions to define attributes as shown in the definition of the attribute ActualRate.

The combination of the propagate symptom statement and one-to-many relationships allow the MODEL language to express complex problem-symptom relationships in a compact form. For example, suppose that there were many multimedia connections layered over the same congested router (possible causing the congestion). In this case, there will be many UDP connections (subclass of TransportConn) layered over the single IP object. The congestion problem may cause symptoms in any or all of the connections which are layered over the IP object.

Now consider trying to write a single rule to express the relationship between the Congestion problem and its symptoms. First, we would have to include complex conditions to identify which multimedia receivers were related to which IP nodes. The MODEL approach of expressing propagation over existing relationships of the object model provides the proper level of abstraction by separating the causal knowledge from the knowledge of the network topology. In addition, by chaining objects together, MODEL can express propagation paths of arbitrary length with ease, while a single rule would require increasing complexity as the propagation paths lengthened.

In addition, the rule language would have to provide some type of *for all* construct, or else there would have to be multiple versions of the rule, one for each possible configuration of multimedia connections over the IP nodes. By breaking the propagation knowledge into discrete units of propagation from a single object to a related object, different topologies at run-time can be handled with a single model. Note however that the main advantage of the rule based paradigm is retained; causal knowledge is expressed in a declarative fashion, independent of the inference engine which uses it.

Up to now, we have focused on multimedia modeling; however, we have been careful to use classes which are not multimedia-specific wherever possible (e.g., IPRouter, TransportConn). This enables us to reuse the invested modeling effort for other applications. To illustrate how MODEL provides for such *modularity of modeling*, we show how to extend our model to a database client domain. This domain will exhibit an entirely different set of symptoms as a result of the congestion at the router (which is a problem *common* to both domains). MODEL allows us to utilize the existing model, and to extend it by adding subclasses and overloading the event propagation in these subclasses to match the behavior of the newly modeled objects.

Database applications typically utilize TCP connections to access database servers. Since TCP connections are reliable, they must retransmit packets which are discarded by underlying IP nodes. Thus the symptom propagation pattern for TCP clients differs somewhat from that of UDP clients. We will use the event overloading capabilities of MODEL to express this difference:

```
interface TCPPort: Port
{
    problem PacketLossHigh =
                ApplicationDelay 1.0, TCPRetransmissionsHigh 1.0;

    propagate symptom ApplicationDelay = Appl, LayeredOver, Delay;
    propagate symptom TCPRetransmissionsHigh =
                TCPConn, PartOf, RetransmissionsHigh;
}

interface TCPConn: TransportConn
{
    readonly intrumented attribute long tcpRetransSegs
                "The total number of segments retransmitted - that \n"
                "is, the number of TCP segments transmitted \n"
                "containing one or more previously transmitted \n"
                "octets.";

    event RetransmissionsHigh = tcpRetransSegs > Threshold;
}

interface DBClient: Appl
{
    problem Delay = TransactionTimeout 1.0, ServerLongLockHolding 1.0;
    propagate symptom ServerLongLockHolding =
                DBServer, ServedBy, LongLockHolding;

    event TransactionTimeout imported;
}
```

Note that TCPPort is derived from Port, but has a different definition for PacketLossHigh than UDPPort, reflecting the different effect packet loss has on a TCP connection. Specifically, the lost packet symptom eventually propagates to the TCP protocol entity which experiences a high rate of retransmission, while the application layered over the node experiences delays; in contrast, the UDP port propagates the lost packet symptom to the application, since it doesn't perform retransmission.

In the case of database clients, the application delay event is further specialized to cause transaction time-outs and long lock holding periods on the server. Note that the event TransactionTimeout is defined as *imported.* This indicates that the event cannot be detected by querying attributes of the data model. Instead an outside entity is responsible for notifying the event correlator of the occurrence of this event. This give maximum flexibility to the modeler to include events which might otherwise be difficult or impossible to monitor.

The event overloading capability of MODEL allows for the creation of very abstract and powerful models, because at each stage of the propagation, the modeler must only concern himself with the immediate effects of a problem on the higher layer. The details of how this effect manifests itself can then be altered by simply deriving a new subclass and refining the definition of events in the subclass. Thus we can express the general notion that congestion at a node causes losses on connections which are layered over the node without having to specify the exact effects of these losses. Subtyping and refinement allow the modeler to specify these effects differently for TCP and UDP connections.

The MODEL language contains many other features which are beyond the scope of this paper. The interested reader is referred to (System Management Arts 1996b).

3 CLASS LIBRARIES IN MODEL

As we have shown, MODEL provides an object oriented modeling framework with inheritance. This makes it ideal for developing extensible class libraries for event modeling. In the examples above, we simply added relationships, attributes and events to the model when needed. In actual MODEL development, we have found that a three stage modeling process works best.

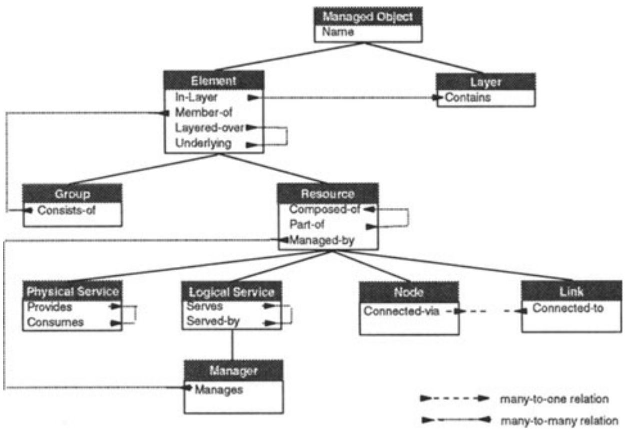

Figure 2 Netmate class hierarchy

In the first stage, a generic library of networking classes is used to define the basic relationships between objects in any modeled system. This set of classes is called the *Netmate hierarchy* is detailed in (Dupuy et al. 1991) and depicted in Figure 2.

The next stage consists of data modeling. Data modeling involves deriving domain specific classes from the Netmate classes and adding the appropriate attribute and instrumentation statements to produce an accurate data model of the domain. In this stage, the *mib2model* translator described above is used to generate class definitions to represent those objects which are instrumented via SNMP MIB's.

The third stage involves adding the actual event propagation information to the model, either directly into the second stage data model, or into subclasses of this model. At this stage, it may be necessary to add additional relationships and attributes to the data model, if it is seen that event propagation occurs over relationships that were not contemplated in the Netmate model, or that important events cannot be monitored in the original data model.

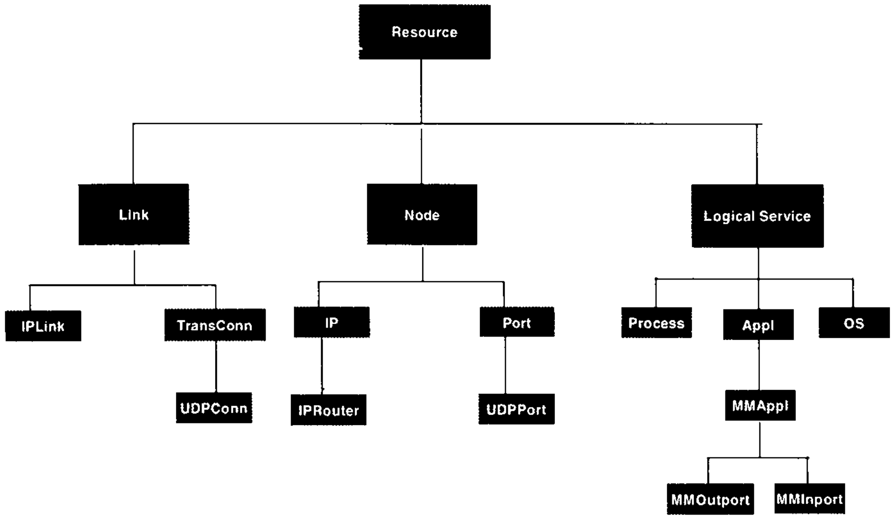

Figure 3 Multimedia Class Hierarchy

Using this methodology, we have developed a Multimedia QoS management library. Figure 3 illustrates the class hierarchy of the Multimedia library. Note that the "root" node is actually the *resource* class of the Netmate class library. The attributes of classes in the library are instrumented via the QoSMIB (Florissi 1996). QoSMIB provides quality of service metrics which are important to diagnosing problems in the multimedia domain. Since QosMIB has an SMI specification and can be accessed via SNMP, we utilized the *mib2model* translator to build a number of the classes in our multimedia library. Consider, for example, the *MM_InPort* class (introduced in section 2) which represents a multimedia receiver. The *MinRate* attribute of this class represents the minimal transfer rate necessary to support the receiving application; this attribute is retrieved automatically from the QoSMIB.

Examples of other domains for which libraries have been developed include problems in the T1 and T3 connections of telecommunications service providers, TCP/IP data networks and low earth orbiting (LEO) satellite networks. These examples illustrate that MODEL provides a basis for developing event libraries for a wide variety of problem domains.

4 COMPARISON TO EXISTING SYSTEMS

In this section, we perform a comparison of MODEL with the event modeling methods of other event correlation systems in the literature. The NetFACT (Houck et. al. 1995) event model has three classes of object: paths, nodes, and shared resources. There are three relationships via which events propagate: Nodes and shared resources have "dependencies" on shared resources; nodes and paths are "connected" to one another; and paths are "composed of" underlying nodes and paths. The NetFACT event model is thus ideally suited for expression in the MODEL language. We have captured the NetFACT event model in about forty lines of MODEL code; space limitations preclude its inclusion in this section.

The NetFACT correlation algorithm involves a voting scheme whereby each symptom event counts as a vote for any problem which may have caused it. This algorithm can be applied to the output of any MODEL language model by simply tracing the propagation backward from symptom to problem. In addition, a MODEL back-end could generate code to automatically tally up votes for each problem via a method generated for each symptom event. Thus, MODEL completely generalizes the NetFACT event model, while giving the users flexibility to add their own new classes, relationships and event propagation rules.

SINERGIA (Brugnoni et. al. 1993) expresses its event model via forward chaining rules which match a particular network topology and use the status of each object in the topology to generate a fault hypothesis for that portion of the network. The generated hypotheses are then fed to a search algorithm which searches for the most likely combination of fault hypotheses. The MODEL event model differs from that of SINERGIA in that instead of specifying particular network topologies and writing rules for each one, the **propagate** statement is used to express the way in which events propagate generally. The expected events for a particular topology can then be generated automatically based on the actual instantiated objects.

SINERGIA's rules closely match the "data sheets" which specify the domain knowledge which is input to the system. Thus, generating MODEL code for SINERGIA would require an additional level of abstraction to be performed. However, if this conversion can be achieved properly, then the resulting MODEL code is more general than the original SINERGIA rules and could be used to generate fault hypotheses for arbitrary topologies. In fact, the SPRINTER event simulator (Manione and Montanari 1995) uses a MODEL-like event propagation model to discover missing and improper rules in the SINERGIA rule base. In addition, writing rules for problems where the events are propagated a very long distance from the problem would seem to be difficult in the SINERGIA methodology, as the size of a SINERGIA rule increases exponentially as the number of components involved. IMPACT (Jakobson and Weissman 1995) also uses a rule-based approach to define when a correlation rule matches the network topology; thus it stands in the same relation to MODEL as SINERGIA.

ECXpert (Nygate 1995) uses rules to define when an incoming event can be correlated with an event or set of events which were previously received. Thus, an

ECXpert rule is similar to a MODEL **propagate** statement, in that it specifies the relationships between events, rather an entire topology of events in a single rule. However ECXpert rules are not as well integrated into the object model as MODEL; thus, ECXpert rules involve string matching to determine event type and database lookup to verify that events have been received from related objects. In addition, ECXpert rules are not completely declarative; the user must specify the rules in terms of an incoming "new event" and the existing "old event" in the context of a particular correlation group to support the correlation algorithm, rather simply providing a relationship between events. In addition, all relationships are defined between alarms; there is no way to specify a problem which itself cannot be observed. Finally, ECXpert requires numbering the events with a precedence indicating which level in the correlation tree the event is expected to occur. This requires one to view the correlation tree as a whole instead of simply providing local propagation rules which expand into a correlation tree based on the current network topology.

5 CONCLUSION

In this paper, we have introduced the MODEL language and showed its application to event modeling. We have shown that MODEL provides a flexible framework for declaratively expressing event propagation which compares favorably to the modeling capabilities of existing systems. Finally, we have also shown how MODEL can be applied develop reusable event libraries and have outlined such a library for multimedia QoS management.

6 REFERENCES

Bolot, J.C., Turletti T., and Wakeman I. (1994) Scalable Feedback Control for Multicast Video Distribution in the Internet. *ACM SIGCOMM 1994.*

Brugnoni S., Bruno G., Manione R., Montariolo E., Paschettra E., and Sisto, L. (1993). An Expert System for Real Time Fault Diagnosis of the Italian Telecommunications Network. In: Hegering, H.-G. and Yemini, Y. (editors). *Third International Symposium on Integrated Network Management,* San Francisco 18-23 April 1993. The Netherlands, North Holland, 617-628.

Busse I., Deffner B., and Schulzrinne, H. (1995) Dynamic QoS Control of Multimedia Applications Based on RTP. *International workshop on high-speed networks and open distributed platforms 1995.*

Dupuy, A., Sengupta, S., Wolfson, O., and Yemini Y. (1991) NetMate: A Network Management Environment. *IEEE Network Magazine.*

Florissi P. (1996). QoSME: QoS Management Environment. Ph.D. Thesis, Columbia University, 1996.

Fry M., Ray, P., Seneviratne, A., and Witana, V. (1996). Multimedia Service Delivery with Guaranteed Quality of Service. *IEEE Network Operations and Management Symposium 1996.*

Houk K., Calo, S., Finkel, A. (1995). Towards a Practical Alarm Correlation System. In: Sethi, A., Raynaud, Y., Faure-Vincent, F. (editors). *Fourth International Symposium on Integrated Network Management,* San Francisco, 1995. London, Chapman & Hall, 226-238.

Jakobson, G. and Weissman, M. (1995). Real-time Telecommunication Network Management: Extending Event Correlation with Temporal Constraints. In: Sethi, A., Raynaud, Y., Faure-Vincent, F. (editors). *Fourth International Symposium on Integrated Network Management,* San Francisco, 1995. London, Chapman & Hall, 290-302.

Kliger, S., Yemini, S., Yemini, Y., Ohsie, D., S. Stolfo (1995) A Coding Approach to Event Correlation. In: Sethi, A., Raynaud, Y., Faure-Vincent, F. (editors). *Fourth International Symposium on Integrated Network Management,* San Francisco, 1995. London, Chapman & Hall, 266-277.

Kumar V. (1996). *MBone, Interactive Multimedia on the Internet.* New Riders.

McCanne, S. and Jacobson V. (1995a). *vic: A Flexible Framework for Packet Video.* ACM Multimedia.

McCanne, S. and Jacobson V. (1995b). *vat: A Visual Audio Tool.* LBL.

Manione, R. and Montanari, F. (1995). Validation and Extension of Fault Management Applications through Environment Simulation. In: Sethi, A., Raynaud, Y., Faure-Vincent, F. (editors). *Fourth International Symposium on Integrated Network Management,* San Francisco, 1995. London, Chapman & Hall, 238-249.

Nygate, Y. (1995). Event correlation using rule and object based techniques. In: Sethi, A., Raynaud, Y., Faure-Vincent, F. (editors). *Fourth International Symposium on Integrated Network Management,* San Francisco, 1995. London, Chapman & Hall, 278-289.

Paxson, V. (1996). End-to-End Routing Behavior in the Internet. In: *ACM SIGCOMM '96.*

Seneviratne, A., Fry, M., Withana, V., Horlait, E. (1994). Quality of Service Management for Distributed Multimedia Applications. In: *IEEE Conference on Computation and Communication 1994.*

System Management Arts. (1996a). *MODEL Language Reference Manual,* White Plains, New York, 1996.

System Management Arts. (1996b). *MODEL Developer's Guide,* White Plains, New York, 1996.

Turletti, T. and Bolot, J.-C. (1994). Issues with multicast distribution in heterogenous packet networks. In: *6th International Workshop on Packet Video.*

Yemini, S., Kliger, S., Mozes, E., Yemini, Y., and Ohsie, D. (1996). High Speed and Robust Event Correlation. *IEEE Communications Magazine,* May 1996.