Composite Events for Network Event Correlation *

G. Liu, A. K. Mok
Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712
{liugt, mok}@cs.utexas.edu

E. J. Yang
SBC Technology Resources, Inc.
9505 Arboretum Blvd.
Austin, Texas 78759
eyang@tri.sbc.com

Abstract

With the increasing complexity of enterprise networks and the Internet, event correlation is playing an increasingly important role in network as well as integrated system management systems. Even though the timing of events often reveals important diagnostic information about event relationships and should therefore be represented in event correlation rules or models, most extant approaches lack a formal mechanism to define complex temporal relationships among correlated events. In this paper, we discuss the formal use of composite events for event correlation and present a composite event specification approach that can precisely express complex timing constraints among correlated event instances, for which efficient compilation and detection algorithms have been developed in [14, 13]. A Java implementation of this approach, called Java Event CorrelaTOR (JECTOR), is described, and some preliminary experimental results of using JECTOR in an experimental network management environment are also discussed in the paper.

Keywords

network management, event correlation, composite event, timing constraint

1. Introduction

In most network management systems, state changes of the *managed objects*, such as link loss, down/up node, (un)reachable remote node, etc., are reported to the *manager objects* in the form of event messages, also called alarms, e.g., SNMP traps or CMIP event notifications. Event occurrences are caused by problems in the network, such as hardware/software failures, performance bottlenecks, configuration inconsistency, security violations, etc., and their effects are observed by the relavant objects in the network. Because a single problem in a networked environment may affect the operation of many devices/subsystems,

*This work is supported in part by a grant from the Office of Naval Research under grant number N00014-98-1-0704.

it may cause a number of events to be generated from them. As a result, network operators are frequently *"flooded"* by a large number of event messages when a few network failures occur. Such scenario is often called *"event storms"* in the network management domain. Because of the large volume of event notifications, these event messages are often ignored or misinterpreted by network operators, and this usually leads to longer latency for problem-diagnosis and bigger loss for businesses that depend on the managed network or system.

Event correlation is an important fault management process to address this problem in many network management systems. By correlating event messages generated from the managed network, a network management system can provide a more concise view of alarm messages such that network operators can accurately and quickly identify the underlying root cause failures. It has been noticed in [8] that "event correlation is becoming one of the central techniques in managing the high volume of event messages". Besides network management, event correlation can also be used in many other mission-critical applications, such as air traffic control, power station and chemical plant management [16], patient-care monitoring, etc., where large volumes of related data are collected from many pieces of equipment and need to be correlated or suppressed for diagnostic purposes. In some emerging integrated system and application management tools, e.g., Computer Associates' TNG Unicenter, Tivoli's TME 10, event correlation is also supported as an important management function.

Event correlation is also a complex task. It involves not only information about the managed network, i.e., topology, configuration, but also knowledge about events generated from various devices/agents in the network, e.g., the format and meaning of each individual event, causal and temporal relationships among events and so on. In both research and commercial contexts, most extant approaches proposed for event correlation, such as model-based [7, 18] and rule-based [1] methods contain various techniques to capture network topology information as well as causal relationships among events. However, timing relationships among events are not well covered in these approaches. In most cases, only a correlation time window is used, which is defined either as a fixed-length window or by a pair of start/end events. As we can see from the following example, it is awkward if not downright impossible to express even some simple cases by a correlation window.

Example 1 *One common event correlation rule in many networks is as follows: "When a link-down event is received, if the next link-up event for the same link is not received within 2 minutes and an alert message has not been generated in the past 5 minutes, then alert the network administrator." These timing requirements cannot be expressed with a simple correlation time window. Instead a conjunction of two conditions with different time windows is required.*

In this paper, we shall present an approach that can specify complex timing constraints among events and show how it can be used for event correlation

in the network management environment. By using this approach, temporal relationships such as those in Example 1 can be defined in a simple and precise way. Efficient algorithms have been developed for compiling and executing the correlation rules thus defined. We have implemented these algorithms; we shall present the results in event correlation and the performance data obtained from some of our experiments.

1.1. Related Work

In the network management domain, a number of approaches have been proposed to correlate alarms/events generated from various parts of the network management system in order to identify the real "cause" of these events, or at least to partially reduce information redundancy. IMPACT [7, 8] adopts a model-based approach, where a network-element class hierarchy is used as the structure model and a message class hierarchy, a correlation class hierarchy and correlation rules are used as the behavior model. NetFACT [6] also has an object-oriented model to describe the connectivity, dependency and containment relationships among network elements. Events are correlated based on these relationships. In EXCpert [15], the *cause-effect* relationships among events are modelled with *correlation tree skeletons* as the bases for correlation. InCharge [18] represents the causal relationships among events with a *causality graph* and uses a *codebook* approach to quickly correlate events to their root causes. However, timing relationships are often ignored or over-simplified in these approaches, important as they are illustrated in Example 1. In most cases, only a correlation time window can be associated with each event correlation process.

In other reported research efforts as well as commercial products, more general AI and other computing techniques have been applied to event correlation. Rule-based system is used in [1]. Neural networks are applied in [17]. In Hewlett Packard's OpenView's Event Correlation Service product, a functional programming language (ECDL) is used to specify correlation rules. In Seagate's NerveCenter product, finite-state machines are used to model the event correlation/fault diagnosis process.

In [5], composite events are proposed for event correlation. The results of correlation, usually the cause events, are specified as composite events, with the correlation rules captured in the specification of these composite events. In this paper, we adopt a similar approach, but the composite event specification/detection approach is very different in our case. In particular, [5] as well as [2, 3, 4] use certain predefined event operators for composite event specification. However, as we indicate in [12], those event operators often introduce ambiguity in terms of composite event occurrence semantics. We have proposed a unified approach for specifying composite events as well as timing constraints; our approach is precise in defining occurrence semantics, and is more expressive and has efficient detection algorithms.

The remainder of the paper is organized as follows: Section 2 introduces our composite event specification approach. Section 3 discusses how we apply this approach to event correlation in the network management environment and describes a prototype implementation of it. Some preliminary experiment results of this event correlation system are shown in Section 4. Section 5 concludes the paper.

2. Composite Events

2.1. Event Model

From an engineering point of view, *events* represent state changes of interests that may occur in a system. For example, "link down", "link up" can be defined as events in a network management system. In general, events can be classified into *primitive events* and *composite events*. *Primitive events* are events that are pre-defined in a system and their detection mechanism is usually embedded in the system implementation. *Composite events* or *complex events* are formed by composing primitive or other composite events, each of which called *component event*. A special module is usually devoted to detecting the occurrences of these composite events.

We also view events, either primitive or composite, as *recurrent*, i.e., an event may occur multiple times during a computation. Each occurrence of an event is called an *event instance*. Since an event may have several attributes associated with it, each instance of it can be represented as a tuple of values of these attributes. Therefore, the *history* of an event is a time-series of these instances, represented as tuples. For simplicity, given an event e, we may also use (e, i) to denote the ith instance of e.

We use the following #-function to define the occurrence index of an event's most recent instance at time t:

Definition 1 *At time t during a computation, $\forall$ event e,*

$$\#(e,t) = \begin{cases} 0 & (e,1) \text{ has not occurred by } t \\ i & e\text{'s most recent instance at time } t \text{ is } (e,i) \end{cases}$$

The following *attribute function* is defined to access the value of attribute *attr* of event instance (e, i):

Definition 2 *Suppose attr is an attribute of an event e. $\forall i \in N^+$, we define the following function*

$$AttrValue(e, attr, i) = \text{the value of attribute } attr \text{ of } (e,i)$$

Furthermore, the following *relative attribute function* is defined to access attribute values of an relative instance of event e at time t:

Definition 3 *Given event e, assume attr is one of its attribute. $\forall$ time t during a computation, $\forall i \in N$, we define*

$$AttrValue_r(e, attr, t, i) = AttrValue(e, attr, \#(e, t) + i)$$

where $\#(e, t) + i > 0$, and t is called the reference time of this function term.

A common attribute of all events is the *occurrence time*. Therefore, we define the following *@-function* and *relative @-function* to represent occurrence time of certain event instances:

$$\begin{array}{lcl} @(e, i) & = & AttrValue(e, \text{“}OccurrenceTime'', i) \\ @_r(e, t, i) & = & AttrValue_r(e, \text{“}OccurrenceTime'', t, i) \end{array}$$

In this paper, the domain of event occurrence time is the set of non-negative integers. Accordingly, the @-function is *monotonic* to occurrence indices, i.e. $\forall i > 0$,

$$@(e, i + 1) > @(e, i)$$

2.2. Composite Event Specification

Most approaches to composite event specification, e.g., [2, 3, 4] use event operators, such as *AND, OR, SEQUENCE*, etc. However, as we pointed out in [9, 12], an event may occur multiple times during a computation, and applying the usual operators on events instead of event instances often leads to semantic ambiguity. This limits the expressiveness of the event specification languages.

We propose a different approach where composite events are specified in terms of conditions or constraints on attribute values of other event terms. From a given set of events, a new composite event can be defined which occurs whenever certain conditions on the attribute values of other event instances becomes true. Furthermore, since the occurrence times of an event obey the *monotonicity* property as indicated in Section 2.1, we have shown in [14, 13] that the truth value of a boolean expression of occurrence attributes can be detected before we evaluate the expression at the occurrences of the component event instances. Therefore, in order to detect composite event occurrences as early as possible, we separate the condition on event occurrences from conditions on other attributes and thereby define a composite event in the following format:

```
define composite event CE with
attributes ([NAME, TYPE], ..., [NAME, TYPE])
which occurs
whenever timing condition
  TC is [satisfied | violated]
if condition
  C is true
then
  ASSIGN VALUES TO CE's ATTRIBUTES;
```

Condition C in the above definition can be expressed as a boolean expression involving *attribute functions* and/or *relative attribute functions*, e.g.

$$AttrValue(e_1, \text{“}attr1\text{''}, 2) + AttrValue(e_2, \text{“}attr2\text{''}, 10) > AttrValue_r(e_3, \text{“}attr4\text{''}, 20, 1)$$

The composite event CE's attribute assignments can be specified in the form *attrname* := *expression*, where *attrname* is the name of an attribute of CE, and *expression* is an arithmetic expression involving *attribute functions* and/or *relative arribute functions* of the component events.

In general, the timing condition TC (which may also be viewed as timing constraint) can be specified as a formula in disjunctive normal forms whose literals are *basic constraints* defined below:

Definition 4 *A* **basic constraint** *specifies a timing relationship between two event instance occurrences. Generally, it can be expressed in the following form:*

$$T_1 + D \ \geq \ T_2$$

where D is an integer constant, and T_1, T_2 *are event instance occurrence time, represented with @-function, relative @-function, or* 0.

Example 2 *The basic constraint* $@(e_1, i) + d \geq @(e_2, i)$ *with* $d > 0$ *specifies the deadline for the ith occurrence of event* e_2 *to be d time units after the ith occurrence of event* e_1. *The basic constraint* $@(e_1, i) - d \geq @(e_2, i)$ *with* $d > 0$ *requires the ith occurrence of event* e_1 *be at least d time units later than the ith occurrence of event* e_2. $@(e, 3) \geq 4$ *means the third instance of event e should not occur earlier than time* 4.

Because our timing constraints and conditions are expressed with respect to event instances, composite events that are specified in this approach have a clear semantics as to their occurrences. Examples of how to express event operators using this formalism are given in [12].

We have developed an event specification language, JESL to support the composite event specification formalism described above. Detailed discussion of the composite event occurrence detection algorithms is out of the scope of this paper. The problem involves detecting the satisfaction or violation of timing constraints and evaluation of attribute conditions and is quite complicated, as has been discussed in [13, 14]. We have developed a set of timing constraint compilation and detection algorithms [13, 14] to detect these composite event occurrences efficiently and as early as possible.

Besides composite events, primitive events can also be defined in JESL in the following syntax:

```
define primitive event PE with
attributes ([NAME, TYPE], ..., [NAME, TYPE]);
```

2.3. Correlation using Composite Events

In the network management environment, we consider those events received at the management stations for correlation as primitive events. These alarms/events are usually generated by the managed network devices (either agents or other software modules on these devices) or by the network management system itself during the polling/monitoring process. Since one or a few faults in the network may cause a large number of these primitive events to occur, a correlation process is used to remove redundant information and identify the root cause of the correlated events. The rules for correlation usually reflect the relationship among the correlated events, such as the causal relationship or temporal relationship. If these relationships can be specified in the composite event definitions, we can view the results of correlation as occurrences of the corresponding composite events.

With the composite event specification language JESL described above, we can express the relationships among events for correlation, either cause-effect or complex temporal relationships as conditions on event attributes for composite events. For instance, for the correlation rule described in Example 1, a composite event LinkADownAlert can be defined as following:

```
define composite event LinkADownAlert with
attributes (["Occurrence Time" : time],
            ["Link Down Time" : time]) which occurs
whenever timing condition
  @(LinkADown, i) + 2 minutes <= @r(LinkAUp, @(LinkADown, i), 1)
  and @(LinkADown, i) + 2 minutes <= @r(LinkADownAlert,
  @(LinkADown, i), 1) and @(LinkADown, i) - 3 minutes >=
  @r(LinkADownAlert, @(LinkADown, i), 0)
  is satisfied
if condition true is true
then {
  "Link Down Time" := @(LinkADown, i);
}
```

where LinkADown and LinkAUp correspond to the up and down events of a link A. The composite event will occur at 2 minutes after an occurrence of LinkADown event if no LinkAUp event occurs during this 2-minute interval and no LinkADownAlert event was triggered during the past 5-minute interval.

Notice in the above example we use LinkADown or LinkAUp to represent the down or up event of link A. However, in most network management systems, only LinkDown and LinkUp events are generated when a link is down or up, and the link name is usually given as an attribute of the LinkDown and LinkUp events. In this case, to facilitate the specification of correlation conditions such

as those in Example 1, we add the following two functions to JESL:

$$\begin{aligned} @_f(e, f, i) &= @(e_f, i) \\ @_{fr}(e, f, T, i) &= @_r(e_f, T, i) \end{aligned}$$

where f is a boolean expression defining conditions on attributes of event e, and e_f is a filtered event based on event e and f such that any instances of event e whose attribute values satisfy f become instances of event e_f. For instance, given LinkDown event, $@(LinkADown, i)$ event in Example 1 can be expressed as $@_f(LinkDown, "Link\ Name" == "A", i)$ where Link Name is an attribute of the event LinkDown that carries the name of the link that is down.

Besides the relationships among correlated events, network event correlation may also involve reasoning about the network topology. For example, a correlation rule may specify that if a router-A-down event is received and many node-down events for downstream nodes from the router A are also received after that, then the root cause is router A down and suppress those node-down events. In this case, although the node-down events may carry the name of the node, whether the node is "downstream from router A" still needs to be inferred from the network topology.

Although our original composite event specification language presented in Section 2.2 does not include conditions on network topology, this can be easily added by allowing predicates in the condition parts of the composite event specification, where predicates can be used to express conditions on network topology. For instance, for the above example, we can use a predicate DownStream(A, B), which is evaluated to true if B is a downstream node from A.

3. JECTOR - A Java Event Correlator

3.1. System Architecture

We have implemented JESL, an event specification language in a Java event service package called JEM. Using JEM, we developd a prototype network event correlation system called JECTOR, implementing the composite event based correlation approach discussed in Section 2.3. The architecture of the JECTOR system is shown in Figure 1.

As shown in Figure 1, JECTOR consists four major components:

- Event Adaptor: Since different network agents or management systems may have different alarm/event message formats, we use *event adaptors* to standardize them into predefined primitive event formats. At run time, the alarm messages are translated into the primitive event instances and fed into the JEM for correlation.
- Event Spec Editor is used for users to input the event correlation rules in the form of composite events. Primitive events, i.e. the events received

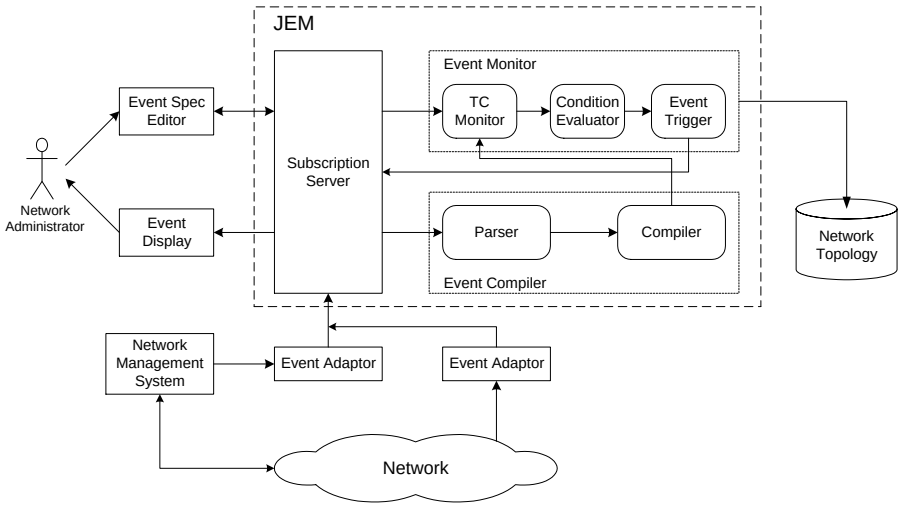

Figure 1. JECTOR System Architecture

from *event adaptors*, may also be specified here. These event specifications, defined with JESL language, are sent to the *event compiler* for parsing and compilation.

- Event Display is used to show network administrators the results of correlation. Occurrences of composite events are received from JEM and displayed here.

- The *event monitor* in the JEM package is used as the correlation engine here. Composite events, which correspond to correlation rules, are compiled and loaded into the *event monitor*. When the network events coming in from the event adaptors, instances of these composite events are detected as the results of correlation. They are sent over to the event display to alert the network administrators. Predicates about network topology, such as *DownStream(A, B)* etc., can be included in the condition parts of composite event specifications, in addition to the event attribute value functions $AttrValue$ and $AttrValue_r$. These predicates are evaluated by the condition evaluator by executing the corresponding built-in procedures according to the network topology data, thus separating the domain-specific from the general event attribute functions. These procedures and the network topology data are customizable such that they can be easily replaced by different implementations for different network environment. The *subscription server* of JEM is used here as a "message switch". All event messages as well as JESL specifications are re-distributed to their relevant components by the *subscription server*.

JEM and JECTOR are implemented in Java, using JDK 1.1.3. The parser for the event specification language is implemented using Java CUP v0.9e, courtesy of a Java LALR parser generator written by Scott Hudson from Georgia Institute of Technology. The event compilation algorithms described in [14, 13] are implemented in the compiler. Currently only an event adaptor for the trap log of HP OpenView Network Node Manager is implemented. It reads the text entries from the HP OpenView's *trapd.log* file and translates them into the predefined primitive event formats. Other types of event adaptors, such as for *syslog*, *SNMP traps*, etc., can be developed if needed.

3.2. Correlation Process

Since we use composite events to represent the correlation rules, the correlation process is essentially the task of composite-event detection through event monitoring. When the primitive event messages are sent from the *event adaptors* and passed to the *event monitor* by the *subscription server*, the timing constraints of pre-defined composite events are first checked against these input event instances by the Timing Constraint Monitor (TC Monitor). If some timing constraints are detected as being satisfied or violated according to the composite event definitions, the condition evaluator is triggered. Here the conditions on other event attributes as well as predicates on network topology are evaluated. A network topology database is used to store the topology data. Once the conditions are evaluated as true, the attribute values of the corresponding composite events are computed and their occurrences are triggered. As a result, the detected composite event instance is sent to *event display* to alert network operators. The new composite event instance may also be fed into the Event Monitor as input for detecting other composite events.

A detailed description of composite event detection algorithms is out of the scope of this paper and can be found in [10]. Here, we use an example scenario to briefly illustrate the detection process for the composite event LinkADownAlert defined in Section 2.3. This scenario and process are described in Table 1.

4. Experimental Results

To evaluate the practicality of our approach, we used JECTOR to correlate events generated from real networks. Our first experiment is conducted on a small IP network test-bed which consists of 7 routers and 50 workstations and PCs. The network is managed by the HP OpenView Node Manager. By studying the event logs of HP OpenView, we found that the most common faults in this network were due to mis-configurations; event messages shown in the OpenView's event windows which correspond to these mis-configurations are usually repetitive and redundant. Network administrators often need to read through hundreds of event messages to identify the problems, which in most cases are only a few. For instance, in the IP network test-bed, some mis-

Event	Index	Occ Time	Detection Operations
LinkADown	1	10:10am	no LinkADownAlert occurred within the last 3 minutes; set timer t_1 to timeout at 10:12am
LinkAUp	1	10:11am	cancel timer t_1
LinkADown	2	10:14am	no LinkADownAlert occurred within the last 3 minutes; set timer t_2 to timeout at 10:16am
LinkADown	3	10:14am	no LinkADownAlert occurred within the last 3 minutes but a time t_2 already set for (LinkADownAlert, 1); no timers set
LinkADownAlert	1	10:16am	timer t_2 timeout; trigger LinkADownAlert event
LinkADown	3	10:18am	(LinkADownAlert, 1) occurred within the last 3 minutes; no timers set
LinkAUp	3	10:19am	do nothing

Table 1. Correlation Process Illustrated

configured subnet masks and IP addresses caused hundreds of event messages in the following format to appear in HP OpenView's event window:

```
...
- Minor Fri Apr 10 09:32:26  x.x.x.x  Inconsistent subnet mask
  255.255.255.224 on interface Et3/0/0, should be 255.255.255.0
- Minor Fri Apr 10 09:34:32  a.a.a.a  Inconsistent subnet mask
  255.255.255.224 on interface Fast, should be 255.255.255.0
...
- Major Fri Apr 10 10:25:05  x.x.x.x  Duplicate IP address: node
  x.x.x.x reported having y.y.y.y, but this address was previously
  detected on node y.y.y.y
...
- Major Fri Apr 10 10:28:04  y.y.y.y  Duplicate IP address: node
  y.y.y.y reported having y.y.y.y, but this address was previously
  detected on node x.x.x.x
...
```

Obviously it is desirable to be able to suppress these repetitive and redundant event messages such that at most one event message, corresponding to one fault (one mis-configuration in this case) is sent to network administrators in a fixed time interval, say 24 hours. Using JECTOR, we can specify two primitive events, InconsistSubnetMask and DuplicateIP, corresponding to these two types of mis-configuration, and then go on to define two composite

events, SuppressedISMask, SuppressedDupIP, such that those two composite events are triggered only if a new inconsistent network mask fault or duplicate IP fault is reported and such fault has not been reported within the previous 24 hours. As a result, we were able to reduce the number of events of these two types from around ***300*** a day to about ***12***. Network administrators can now quickly find the problems and fix them. And it also becomes easier for them to find other types of errors which would otherwise be "buried" in these event message "flood". JECTOR's response time in these experiments, i.e., the detection latency of the composite events is under 1 second, and this is deemed to be quite acceptable for most practical scenarios today.

Scalability is an often asked question when a system's performance is under evaluations. In our case, the composite event detection latency D is the most important indicator of JECTOR's performance. Although our experiments show acceptable detection latencies in the test cases, D could go up dramatically when the number of composite events to be detected or the event occurrence rates go beyond certain thresholds. A separate set of experiments were conducted to measure D under different event occurrence rates. The results, which are discussed in detail in [11], show that D is largely determined by the balance between event occurrence rate and the processing time of each event instance. If the event processing speed cannot catch up with the event occurrence rate, the wait queue will keep getting longer and eventually lead to unacceptably long latencies for composite event detection. For the composite event set used in the above network event correlation experiments, our tests show that D is within 1 second, as long as the event occurrence rate does not exceed 100 per second.

There is room for improving the performance of event detection under high event rates, by reducing the average event processing time and thereby decreasing the length of the wait queue. Some optimizations have been implemented in the current JEM system such that conditions shared by several composite events need to be checked only once instead of multiple times. This would help reduce the average event process time when a condition is shared in multiple composite event specifications. Other techniques may include pipelining event processing and load-sharing on multiprocessors. By pipelining the composite event detection process, we can improve the average event processing speed and thus have shorter wait queues. If the composite events and other involved component events can be partitioned into several independent sets, we can assign them to several event correlators running on different processes or machines such that each of them may experience smaller event reception rate and less event processing time.

5. Conclusions and Future Work

In this paper, we have described an approach for event correlation which is based on a formal composite-event specification language and which can precisely express timing relationship among correlated event instances. By applying this approach to a network management environment testbed, using a Java package called Java Event Correlator(JECTOR) which implements this approach, we seeked to demonstrate the viability of our correlation method in practice. The performance results of JECTOR in some experiments, in particular, event correlation latency are also discussed in the paper.

The intentional separation of network topology model and reasoning from the event specification and detection mechanism may be regarded by some as a disadvantage of this approach, compared to model-based event correlation approaches. However, we think that this flexibility is what makes it possible to apply this approach to different network topology model as well as other management domains. Our goal is to have a modular and composable toolset which is consistent with the open architecture philosophy which is extremely important consideration in our opinion. For our future work, besides working on techniques to further reduce detection latency under heavy load (i.e., large event rates), we plan to integrate JECTOR more closely with HP OpenView Node Manager, since it is a widely used network management platform today. And we shall also seek to use JECTOR in a larger scale network environment to further test its practicality and identify new research problems.

References

[1] S. Brugbosi, G. Bruno, and et al. An expert system for real-time fault diagnosis of the italian telecommunications network. In *the 3rd International Symposium on Integrated Network Management*, pages 617–628, 1993.

[2] S. Chakravarthy and D. Mishra. Snoop: An expressive event specification language for active databases. *Data and Knowledge Engineering*, 14(10):1–26, October 1994.

[3] S. Gatziu, A. Geppert, and K.R. Dittrich. Detecting composite events in active database systems using petri nets. In *the 4th International Workshop on Research Issues in Data Engineering*, pages 2–9, Feburary 1994.

[4] N. Gehani, H.V. Jagadish, and O. Shmueli. Composite event specification in active databases: Model & implementation. In *the 18th International Conference on Very Large Data Bases*, pages 327–338, August 1992.

[5] M. Hasan. *The Management of Data, Events, and Information Presentation for Network Management*. PhD thesis, Department of Computer Science, University of Waterloo, Canada, 1995.

[6] K. Houck, S. Calo, and A. Finkel. Towards a practical alarm correlation system. In *the 4th IEEE/IFIP Symposium on Integrated Network Management*, 1995.

[7] G. Jakobson and M. Weissman. Alarm correlation. *IEEE Network*, 7(6), November 1993.

[8] G. Jakobson and M. Weissman. Real-time telecommunication network management: Extending event correlation with temporal constraints. In *the 4th IEEE/IFIP Symposium on Integrated Network Management*, 1995.

[9] Guangtian Liu and Aloysius K. Mok. An event service framework for distributed real-time systems. In *IEEE Workshop on Middleware for Distributed Real-Time Systems and Services*, December 1997.

[10] Guangtian Liu and Aloysius K. Mok. An event service framework for distributed real-time systems. Technical report, Real-Time System Lab, Computer Science Department, University of Texas at Austin, June 1997.

[11] Guangtian Liu and Aloysius K. Mok. Implementation of jem - a java composite event package. Technical report, Real-Time System Lab, Computer Science Department, University of Texas at Austin, November 1998.

[12] Guangtian Liu, Aloysius K. Mok, and P. Konana. A unified approach for specifying timing constraints and composite events in active real-time database systems. In *Real-Time Technology and Applications Symposium*, June 1998.

[13] Aloysius K. Mok and Gunagtian Liu. Early detection of timing constraint violation at runtime. In *Real-Time Systems Symposium*, December 1997.

[14] Aloysius K. Mok and Gunagtian Liu. Efficient run-time monitoring of timing constraints. In *Real-Time Technology and Applications Symposium*, June 1997.

[15] Y.A. Nygate. Event correlation using rule and object based techniques. In *the 4th IFIP Symposium on Integrated Network Management*, 1995.

[16] M. PlauWagenbauer and W. Nejdl. Integrating model-based and heuristic features in a real-time expert system. *IEEE Expert Intelligent Sys. and their Applications*, 8(4):1218, 1993.

[17] H. Wietgrefe, K. Tochs, and et al. Using neural networks for alarm correlation in cellular phone networks. In *the International Workshop on Applications of Neural Networks in Telecommunications*, 1997.

[18] S. Yemini, S. Kliger, and et al. High speed and robust event correlation. *IEEE Communications*, May 1996.